FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Mohammed Bashar Husham Al-Ani

RESTFUL WEB SERVICES FOR AN ERP SYSTEM FOR SOCIAL SERVICES

Master's Thesis
Degree Programme in Computer Science and Engineering
2019

Al-Ani M. (2019) RESTful Web Services for an ERP System For Social Services. University of Oulu, Degree Programme in Computer Science and Engineering (Ubiquitous Computing). Master's Thesis, 56 p.

ABSTRACT

The advances in hardware and software have been rapidly integrated by organizations, especially in the healthcare sector, demanding new approaches for software to provide more reliable products, under well-known quality standards. This thesis investigates, designs and implements a set of operationally crucial RESTful web services for Invian Oy ERP system, DomaCare. Today, DomaCare is one of the fastest growing and developing software solutions in Finland in the healthcare sector. Thousands of satisfied healthcare professionals across Finland use DomaCare daily. DomaCare is a versatile ERP system designed specifically for the social sectors.

This thesis describes the theoretical part of software architecture and software architectural style, which support understanding REST. Second, the thesis introduces the environment and tools required for the development stage. Third, the thesis presents the action and sequence diagrams for each use case to support the overall understanding of the system in a higher level of abstraction. Moreover, unit tests were implemented in this thesis for each use case, and, finally, the approach which was employed to validate the system is presented.

In conclusion, the thesis concludes that based on the literature review, implementation, the results obtained from the unit tests, and the system validation fulfilled the goals set for this thesis.

Keywords: ERP, RESTful, REST, design, implementation.

Al-Ani M. (2019) RESTful Web Services For an ERP System for Social Services. Oulun yliopisto, tietotekniikan tutkinto-ohjelma (Ubiquitous Computing). Diplomityö, 56 s.

TIIVISTELMÄ

Laitteistojen ja ohjelmistojen tekniikkaa on nopeasti integroitu organisaatioihin erityisesti terveydenhoitoalalla. Tämä vaatii ohjelmistojen osalta uudenlaisia lähestymistapoja, jotta jatkossa voidaan tarjota luotettavampia ja tunnettujen laatustandardien mukaisia tuotteita. Tässä diplomityössä esittelen tutkimusvaihetta, suunnitteluprosessia sekä toteutustapoja toiminnallisesti tärkeän, REST-arkkitehtuurimallia noudattavan webrajapinnan toteuttamiseksi Invian Oy:n DomaCare-toiminnanohjausjärjestelmässä. Tänä päivänä DomaCare on yksi Suomen terveydenhuoltoalan nopeimmin kasvavista ja kehittyvistä ohjelmistoratkaisuista. Sitä käyttävätkin tuhannet tyytyväiset terveydenhuollon ammattilaiset päivittäin ympäri Suomen. DomaCare on monipuolinen asiakas- ja toiminnanohjausjärjestelmä, joka on suunniteltu erityisesti sosiaalialoille.

Diplomityössäni kuvaan verkkopohjaisten järjestelmien arkkitehtuurisuunnittelun sekä REST-arkkitehtuurimallin teoreettista puolta. Lisäksi tuon opinnäytetyössäni esille kehitystyöhön tarvittavan ympäristön ja työkalut sekä toiminta- ja sekvenssikaaviot kullekin käyttötapaukselle, jotta se tukisi yleistä järjestelmän ymmärtämistä korkeammalla abstraktiotasolla. Lisäksi esittelen tutkimuksessani kaikki toteutetut yksikkötestit kussakin käyttötapauksessa sekä lopuksi myös ne lähestymistavat, joita käytettiin systeemin vahvistamiseksi.

Päädyn johtopäätökseen, että diplomityössäni esitellyt kirjallisuuskatsaus, toteutus ja yksiköiden testeistä saadut tulokset täyttivät tämän tutkimuksen tavoitteet.

Avainsanat: ERP, RESTful, REST, design, implementation.

TABLE OF CONTENTS

FOREWORD

This thesis has accomplished in Invian Oy and the center of Ubiquitous computing, University of Oulu.

Foremost, I would like to thank my supervisor, Simo Hosio, for his support and dedication for reviewing my master thesis. He gave precious advises and guidelines. I want to thank my second supervisor as well from Invian Oy, Arttu Tanner, for his support, patient and approving my work. A special thanks to my girlfriend, Ulla Siira, who supported me from the beginning and kept my motivation and morale high. Tremendous thanks to my best friends, Ibraheim Al-Khudairi and Ahmed Hamid, for giving me support during my journey from brainstorming to overcome some significant obstacles that I faced. Finally, my special gratitude goes to my family for their love, support, and inspiration, as well to all my relatives and my friends who supported me.

Oulu, 21.03.2019

Mohammed Al-Ani

ABBREVIATIONS

WS	Web Services
REST	Representational State Transfer
ERP	Enterprise Resource Planning
SOAP	Simple Object Access Protocol
WSDL	Web Services Definition Language
HTTP	Hypertext Transfer Protocol
CRUD	Create, Retrieve, Update and Delete
URI	Unified Resource Identifier
JSON	JavaScript Object Notation
XML	Extensible Markup Language
SQL	Structured Query Language
JOOQ	Java Object Oriented Querying
OOP	Object Oriented Programming
POJO	Plain Old Java Object
ORM	Object Relational Mapping
DAO	Data Access Object
JDK	Java Development Kit
JRE	Java Runtime Environment
API	Application Programmable Interface
ER	Entity Relation
UML	Unified Modeling Language
JSR	Java Specification Request
DB	Database

1. INTRODUCTION

DomaCare is a versatile ERP system designed and developed specifically for the needs of social sectors. ERP has improved considerably in the past decade and is now even more evolving than ever before. Previous decade has seen enterprise resource planning (ERP) solutions improve and evolve into versatile and multi-faceted tools that help small, medium and large organizations to run their business. The ERP evolution was not only about changing the technology but also to have different priorities for users which needed to address in one smart solution.

Simply, DomaCare is a product application used in social sector services such as home services, rehabilitation, childcare and mental health. In home services, DomaCare does the work in one click using modern web user interface.

In DomaCare the billing is completely automated, the powerful feature in DomaCare is that it has the billing system built in it. The billing section built very carefully and with knowledge about the field. DomaCare has more than 30 integrations to different financial management systems like Netvisor, Passeli, Netbaron etc. The target behind the billing system is to make the monthly billing processes easier for the company to manage their billings. The most up-to-date information is always in use by a worker in need of it. The data stored for customers guaranteed to be securely stored servers that backed up at regular intervals.

Therefore, DomaCare considered a complete ERP solution as it has services that provide base functionalities that forms a main part of the requirement of ERP system nowadays.

1.1. Purpose and Objective

The purpose of this thesis is to implement production-ready RESTful web services that will accomplish the requirement of Invian Oy ERP for resource planning for social healthcare services. Therefore, the primary objective of this thesis is to design and implements RESTful API for resource management ERP system. The system must fulfill all the following requirements which are the basic to run the ERP.

1. To design RESTful API according to the REST architectural style.
    a. The existence of client-server components.
    b. APIs requests must be stateless.
    c. The communication between client-server components must happen through uniforms interfaces.
2. Understanding the resource planning database relations and provide a visual image of how these tables related in practice.
3. Studying the customer requirements carefully and come up with a solution describing it in a graphical diagram.
4. The Implementation of the REST APIs should do according to the requirements.
5. Documenting the REST APIs.
6. Testing the implemented system.
7. Validating the implemented system.

1.2. Scope

This thesis will study in detail the literature review including but not limited to REST, database tables relationship and some other tools that will use during the development stage. This thesis will first start describing the system in higher abstraction, and activity diagrams will be proposed based on the understanding of customer requirements. After obtaining the activity diagrams, this thesis will implement the REST APIs, services and DAOs. Then model the API through implementing the sequence diagrams for an end to end use case. Moreover, documenting all RESTful APIs, and finally testing the APIs using the unit tests and validating the APIs in quality assurance and production environment with one of Invian Oy clients.

In summary, the theoretical part will be studied in detail in the next chapter to support the development of REST API later. Chapter 3, discuss detailed information about the environment and tools used, and then the implementation part will be discussed in detail with many samples of UML, activity diagrams, data representations, and codes. In chapter 4, presenting unit tests for each case in the system with the results of the tests, and how the implemented system will be validated. Chapter 5 presents the discussion of the achievement of the thesis goals and improvement of Invian Oy ERP system. Finally, in chapter 6, shows the conclusion of the thesis with the result gained from the development.

2. RELATED WORK

In this chapter, the theoretical part will be introduced. This thesis will begin with the term's definitions of software architecture and software architectural style as they are later necessary for a better understanding of REST. Then this thesis will study step by the step the move toward REST which includes an introduction about Fielding approaches REST as an architectural style, constraints, and elements description that compose REST, the higher abstraction of the RESTful system and explaining the REST system over HTTP.

As a brief introduction, REST came from Representational State Transfer and was introduced by Roy Thomas Fielding in his Ph.D. dissertation: Architectural Styles and the Design of Network-based Software Architectures." published in 2000. REST is not a software architecture by itself, but a coordinated set of architectural constraints which tries to minimize latency and network communication, while maximizing the independence and scalability of component implementations [1].

2.1. Software Architecture

There are many issues in the past when mentioning software architecture; one of these issues is the term of the software architecture used so much by different authors but in an inconsistent way. Fielding created his definition based on previous researches like [2].

Software architecture defined by a configuration of architectural elements—components, connectors, and data—constrained in their relationships to achieve the desired set of structural properties [3].

There is a standard definition from ISO that makes the concept more generally. Based on ISO/IEC/IEEE, architecture which is referring to system or software-defined as the fundamental concepts or properties of a system in it is environment embodied in it is elements, relationships, and in the principles of it is design and evolution [4].

Based on the previous definitions it can conclude that software architecture is a high-level of an abstraction of a system which is focusing on the visibility of elements in components, connectors, and properties, the last two are forming the relationships between components.

2.1.1. Components

Component is a collection of the computational elements together accompanied with a description of the interactions between these components-connectors [5]. Computational elements are the abstract unit of software instructions, and internal states that provide a transformation of data via it is interface [3]. Client-server and database are examples of components.

2.1.2. Connectors

Connectors described as the glue that contains and holds all the architecture components together [2]. An accurate definition of a connector defined as an abstract

mechanism that mediates communication, coordination, or cooperation among elements [6]. Implementation of the architecture contains the information of it is components and connectors which hidden at the architectural level. RPC or any communication protocols like client-server consider being an example of connectors.

2.1.3. Data Elements

Data elements include the information that is used and transformed [2]. “A datum is a single element that has information which is transferred from a component, or received by a component, via a connector” [3]. Properties are the constraints that define a set of conditions and restrictions between components. One of the first definitions of features related to software architecture was a composition between properties, which are used to define constraints on the elements and relationships, which are used to constrain how the different parts may interact and how they are organized to each other in the architecture [2].

In summary, properties are extra information about the elements and their associated relations [7]. The Properties induced by the set of constraints within an architecture [3].

2.2. Software Architecture Style

An architectural style is the definition of rules that define a set of rules that describe how the component interacts. Also, it identified as a specialization of element and relation types, together with a set of constraints on how they can be used [7]. The definition is close as the software architecture but notes the term style versus a high level of abstraction. Therefore, an architectural style can be though as a set of constraints applied to architecture.

The main thing about an architectural style is that it wraps the necessary decisions concerning the structural elements and emphasizes essential constraints on the parts and it is relationships [7].

2.3. REST Approach

REST presented in Roy T. Fielding dissertation [3]. Therefore, a study to the software architecture and style was mandatory and considered to be the heart of REST.

REST described as an architectural style for distributed systems. REST is a hybrid style which comes from different network architectural style and with the extra set of rules that will define a uniform connector interface. REST constraint on constraints that must be presented and put on the connector [3].

Fielding started from the beginning where identifying the system needs without constraints and slowly step by step adding those to the system. Some additional restrictions belong to REST which comes from the architectural style which includes client-stateless-server, stateless-server, uniform interface, cacheable, client-server and layered-system [3].

2.4. Constraints of RESTful Architecture

2.4.1. Client-Server

In the client-server interface, there must be a client component exists to send requests to server component which will receive the client requests and process that and respond with a response. The uniform interfaces main responsibility is to separate the client and server interfaces meaning that the client does not concern about the data storage which is the server responsibility and same for the server not concerned about the user interface which is the client responsibility. With that said, the separation improved portability of these interfaces to cross-platform and improved the scalability of server components. This separation as well helped the segregation of client logic from the server logic as they can develop independently with a condition of the uniform interface must not change [3].

2.4.2. Stateless

This constraint added to client-server for every interaction, the client-server communication must be stateless. Stateless means the client request must hold all the required information for the server to process the request. The server side does not store any information about the session and client info instead of the session information stored at the client side [3].

2.4.3. Cacheable

Cacheable constraints will assist with a reduction in network performance. The addition of cacheable restrictions will reduce the requests being processed at the server side if the required information already for some HTTP requests exists in the local cache. Adding the cache constraints to REST brought some advantages like improving the efficiency but with drawback which is the reliability could be an issue if the data obtained from the local cache is different from the one from the server side [3].

2.4.4. Uniform Interface

The main feature that distinguishes the REST architectural style from other network-based form is that emphasis on a uniform interface between components. By applying the generality to the component interface, the overall system architecture simplified, and the visibility of interactions is improved. The uniform interface degrades efficiency since information transferred in a standardized form rather than one which is specific to an application need. With, in-order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components [3].

2.4.5. Layered System

The layered system allows software architecture to be composed of hierarchical layers by containing component behavior such that each REST component cannot "see" beyond the immediate layer with which they are interacting. Layers can be used to wrap legacy services and to protect new services from legacy clients, simplifying component by moving infrequently used functionality to a shared intermediary. The intermediary can improve the system scalability by enabling the load balancing of services across multiple network and processors [3].

The main disadvantage of layered systems is that they add overhead and latency to the processing of data, reducing user-perceived performance. Placing a cache between layers can improve performance [3].

2.4.6. Code on Demand

The last set of REST constraints is the code on demand. REST allows client functionality to be extended in away of downloading code from the server and executing it in the form of scripts or applets. The advantage of this constraint is to minimize the numbers of features that required to pre-implemented. The disadvantage of this is reducing the visibility, and therefore this constraint included as an optional [3].

2.5. REST Elements

The REST architectural style designed to fit in stateless communication which typically is the HTTP. Clients-Servers are exchanging representations of resources by standardized uniform interfaces. "REST neglects the details of component implementation and protocol syntax to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements" [3].

2.5.1. REST Components

The REST component summarized in the below table.

Table 1. REST Components [3]

Component	**Modern Web Examples**
Origin Server	Apache httpd, Microsoft IIS
Gateway	Squid, CGI, Reverse Proxy
Proxy	CERN Proxy, Netscape Proxy, Gauntlet
User Agent	Netscape Navigator, Lynx, MOMspider

User-agent uses a REST client connector to initiate a request and becomes the ultimate recipient of the response. An origin server uses a server connector to govern the namespace for a requested resource [3]. A proxy is an intermediary selected by a client to provide interface encapsulation of other services, data translation, performance enhancement or security protection. A gateway is an intermediary

imposed by the network or origin server to provide an interface encapsulation of other services for data translation, performance enhancement, or security protection [3].

2.5.2. REST Connectors

The REST connectors present an abstract interface for component communication, enhancing simplicity by providing a clean separation of concerns and hiding the underlying implementation of resources and communication mechanisms. [3]. REST uses different types of connectors and summarized in table 2.

Table 2. REST Connectors [3]

Connector	**Modern Web Examples**
Client	libwww, libwww-perl
Server	libwww, Apache API, NSAPI
Cache	browser cache, Akamai cache network
Resolver	bind (DNS lookup library)
Tunnel	SOCKS, SSL after HTTP CONNECT

Client and server are the primary connector types. The client initiates the communication by requesting whereas the server is listening for incoming requests and responds requests for supplying access to it is services. A cache can be located at the interface of the client or the server connector to save cacheable responses to current intercourse so that they could reuse for later requests. A resolver translates partial or complete resource identifiers into the network address information needed to establish an inter-component connection [3]. The use of one or more resolver adds request latency but on another hand can improve the longevity of the resource's references. Finally, the tunnel type connector which is simple relays communication across a connection such as a firewall or low-level network gateway [3].

2.5.3. Data Elements

A distributed hypermedia architect has only three fundamental options which first render the data where it located and send a fixed format image to recipient, second is to encapsulate the data with a rendering engine and send to recipient and third is to send raw data to the recipient along with the metadata that describes the data type [3]. REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types selected dynamically based on the capabilities or desires of the recipient and the nature of the resource [3]. The REST data elements summarized in table 3.

The resource is the fundamental abstraction of information in REST. Any resource object that possible to have a name can be called a resource. A resource object is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any point in time. REST uses a unique identifier to identify the resource involved in an interaction between components. The connectors provide a generic interface for accessing and manipulating the value set of a resource object. A uniform resource identifier (URI) is an array of characters used to identify a name of the resource. Such identification enables interaction with representations of the

resource entity over a network [8]. Representation response of resources is what send back and forth between components. The representation response, in general, is a binary stream that holds the metadata which describes how it must consume. Resource metadata is the information about the resource which is not specific to the supplied representation. Control data defines the goal of the message between the REST components.

Table 3. REST Data Elements [3]

Data Element	**Modern Web Examples**
Resource	The intended conceptual target of a hypertext reference
Resource Identifier	URL, URN
Representation	HTML document, JPEG image
Representation metadata	Media type, last-modified time
Resource metadata	Source link, alternates, vary
Control data	If-modified-since, cache-control

2.6. REST API

API stands for Application Programming Interface. An API is a set of Functions that can do one or more tasks to be used by other software. RESTful systems communicate with the HTTP. REST API is a library based implemented within the HTTP standard.

2.7. REST through HTTP

HTTP is a stateless application-level protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems [9].

Communicating the representation between client and server is done via HTTP. In RESTful there are four main functions which used for persistence purposes, CRUD (Create, Read, Update, Delete), to be set to the resources. In HTTP definition standard, the CRUD can be translated into HTTP methods which are POST, GET, PUT and DELETE. Other HTTP methods can be used such as OPTIONS, TRACE, and PATCH. HTTP methods are part of uniform interfaces because they provide action based on HTTP verb.

- The GET method is used to query data (representation) of a resource.
- The POST method is used to create new resources (entities).
- The PUT method is used to modify a single existing resource(entity).
- The DELETE method is used to remove a unique resource(id).

3. DESIGN AND IMPLEMENTATION OF DOMACARE RESTFUL APIS

3.1. Introduction

The primary goal of this research is to Develop RESTful services for an ERP System for social services. In chapter two, this thesis discussed the background of REST Architecture and Design. This chapter explains how to develop and implement REST API for an ERP system using Java with Jersey framework [13].

This thesis uses the already implemented database structure in Invian Oy, which used for the resource-planning module as a base. Therefore, this thesis starts from this base to study the database relations and modal these relations in a visual view like diagrams to be used later for designing and implementing the RESTful web services.

3.2. Environment and Tools

The primary coding language used for developing REST API in Java. Java is a high-level object-oriented programming language for cross-platform development. Java is very close to human being, and it is easy for humans to code with Java other than machine language. Java is one of the most famous programming languages used for developing client-server application and applets. Thus, the compiled Java program can run in every machine that supports JVM (java virtual machine) [10].

To develop an application with Java programming language, java development kit JDK must installed. The JDK shipped as well with java runtime environment JRE to run the java program. The REST framework which used is Jersey. Jersey framework is open source and production ready for developing RESTful Web Services in Java that uses JAX-RS APIs. Jersey considered as the reference implementation of JAX-RS specification(JSR 311, JSR 339) [13].

Maven used as build and dependency management tool, Maven, is a software project that bases on the concept of project object model POM. It is excellent for managing project dependencies, build and documentation. Maven has a central repository. Maven has a build cycle and phases that useful for validation and testing.

IntelliJ idea integrated development environment (IDE) used for coding and compiling the source code. The idea has a high integration with other tools like GIT and Maven and Java enterprise. The ultimate version was used to give full integration support to enterprise development tools [17].

Git is a distributed version control system to keep tracking changes in the source code. Git is an excellent tool for every programmer as it allows to commit new changes and revert changes and merge changes.

Tomcat is one of the standard Java servlet containers, it is open source, and it implemented some of the Java enterprise specifications like java servlet, web sockets, java server pages, and java expression language. Tomcat provides a pure Java HTTP web server that can run a java program. MySQL used for storing data in the relational database. It is an open source and free to use under general public license GNU. Many web application and popular websites use MySQL. MySQL is mighty and efficient. It uses SQL structured query language for dealing with all kind of CRUD create, read, update and delete operations [14].

JOOQ is a type-safe, object-oriented API for integrating SQL language in a way that allows the developer to write typed safe queries. The benefit of using JOOQ is to drive the focus on domain business logic [16].

3.3. Data Format

JavaScript object notation JSON as lightweight data format [11]. JSON is easy to read by human cause there are not those ending tags like the ones in XML. Both JSON and XML is human-friendly but, in this project, we use JSON as the base format. JSON is easy to parse and generate. There are many Java libraries for generating JSON from java object and vice versa. One of the most popular libraries is Jackson. Please see figure 1 JSON data sample.


```
{
    "id": 6718533411,
    "name": "2007 ford mustang",
    "make": null,
    "model": null,
    "price": 5500,
    "linkToItem": "https://newyork.craigslist.org/que/cto/d/2007-ford-mustang/6718533411.html",
    "linksToImages": [
        {
            "id": null,
            "itemId": 6718533411,
            "url": "https://images.craigslist.org/00e0e_cX0EG33wUKV_600x450.jpg"
        },
        {
            "id": null,
            "itemId": 6718533411,
            "url": "https://images.craigslist.org/00e0e_5d8M4mOFmql_600x450.jpg"
        },
        {
            "id": null,
            "itemId": 6718533411,
            "url": "https://images.craigslist.org/00L0L_lxcoXhvb4Nl_600x450.jpg"
        },
        {
            "id": null,
            "itemId": 6718533411,
            "url": "https://images.craigslist.org/01010_hWn7jcDowA1_600x450.jpg"
        }
    ]
}
```


Figure 1. JSON Sample.

3.4. Implementation

The target is to create RESTful API for Invian Oy ERP system for social health care. There are two primary parts in the system which are the employees and customers. The employee is the user who is using the system, and the employee has different privileges to access APIs and customers data. This thesis will discuss two roles which are admin and user roles. Admin role can do all kind of tasks which can be extended from creating customer resource templates and to planning them in the future, admin role can assign customer tasks to employees who can accomplish the task. User role has fewer privileges than admin role, and user role can list all the customer tasks assigned to him/her, user role must not be able to see duties which are not attached to him/her, user role can process customer task and realize them once are done. In short, here is a list of the requirements.

- Admins can plan resources in future (repeated).
- Admins can link billable products to resources.
- Admins can update resources base templates.
- Admins can not update resource products belongs to customer template.
- Employees can read only the assigned tasks.
- Employees can link additional products if needed to customer tasks.

- Employees can cancel tasks.
- Employees can start tasks.
- Employees can finish tasks.
- Privilege system.
    - Employees can view only items assigned to them.
    - Employees can modify objects only attached to them.
- Completing customer tasks must take care of realizing billable products.
- Finished customer tasks are unmodifiable.

3.4.1. Entity Relations

Entity relations diagram is a graphical representation of entities and their relationships to each other. Entity relations model is useful at the initial step of each project, and entity relation knowledge will assist on avoiding bad design even that is not the case with thesis, a study of the current database relation is a must. Studying the entities relation will open ups opportunities for improvement in this thesis and the future.

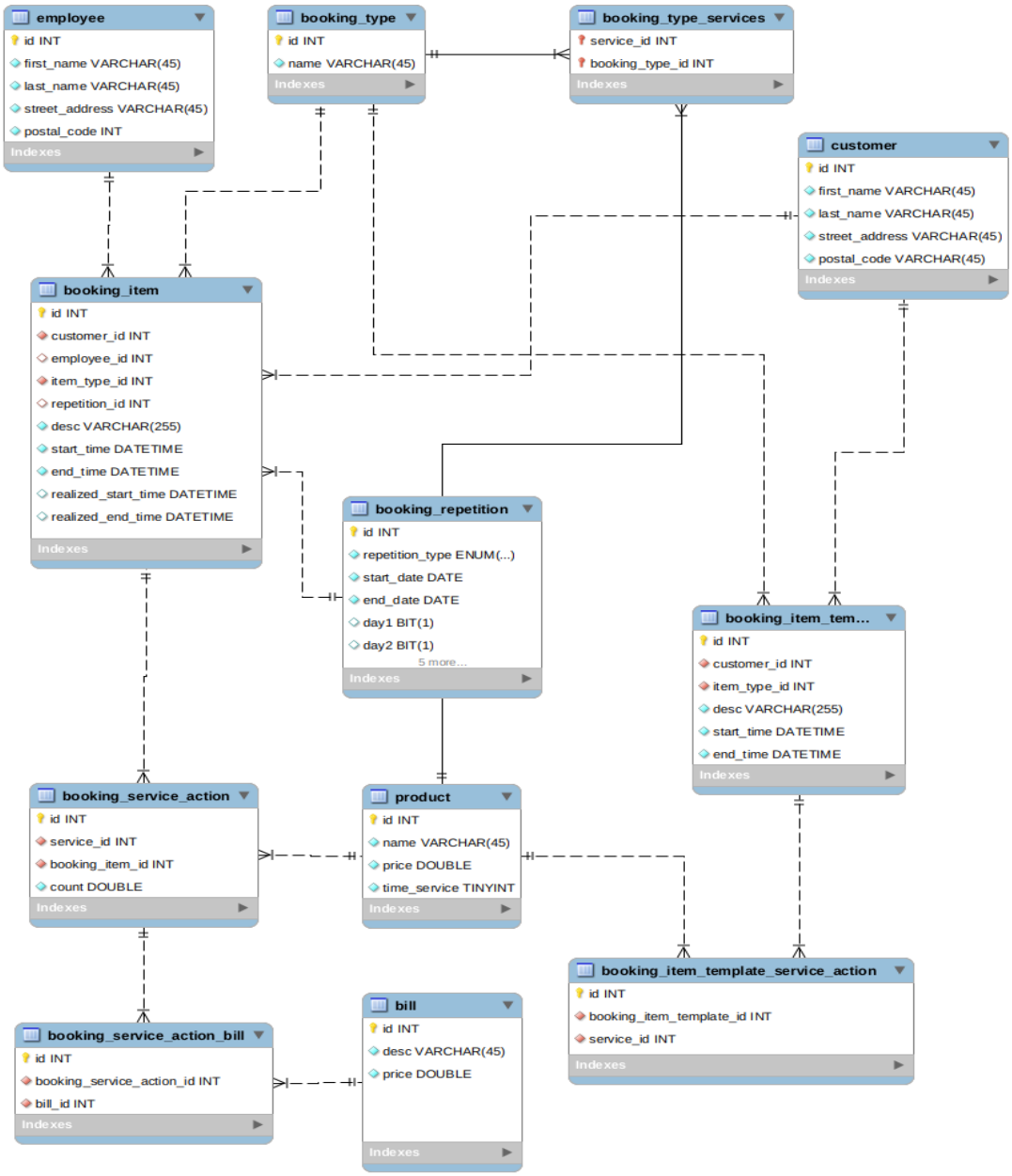


Figure 3. ER diagram.

Figure 3 shows how are entities are related, customer and employee entities both have common attributes like id, first name, last name, street address and postal code the identifier id is a primary key that identifies the customer and employee entities.

Booking item entity has attributes of id, customer_id, employee_id, start time, end time, realized_start_time, realized_end_time, item state. Booking item id is the primary key which identifies the booking item entity. The customer_id is foreign key FK to the customer entity, and the employee_id is FK to employee entity, repetition_id is FK to booking repetition entity. The start time and end time define the planned times specified by the admin for this particular entity. The relalized_start_time and realized_end_time defining the actual time when the booking item being processed and ended by the employee.

Booking _item template is defining the customer template, and the template purpose is to use for planning booking item in the future, the model has the same

attributes as the booking item entity except that there is no employee_id attribute and that makes sense as it meant for creating booking item which not assigned to any employees. Booking repetition is defining how these booking items will plan in a calendar. There are a set of different types that could apply to customer resource template like (every day, every week, every-n-days). Below are how these entities are related.

- Customer entity can have many booking items.
- Employee entity can have many booking items.
- Customer entity can have many booking item templates.
- Booking item can belong to one customer and customer is mandatory.
- Booking item template can belong to one customer and customer is compulsory.
- Booking item can have many products.
- Products can have many booking items
- Booking service action entity is a link of many to many between booking item and product entities.
- Bill can have many products.
- Booking service action can have many bills.
- Booking service action bill entity is a link of many to many between booking service action and bill entities.

3.4.2. POJOs Submodule

After understanding the ER, now it is the time to generate the entities. There are many ways of generating entities.

- Manually create one by one.
- Using JOOQ POJOs generator to generate JOOQ classes.
- Using hibernate to produce from the schema.

3.4.3. Rest Submodule

The REST resources are responsible for handling the incoming HTTP requests, and each HTTP request will have an HTTP method. The HTTP request can have a payload and set of constraints. Once the request received by the resource process, the received request will start and once the process operation done then return the result of the processed request and wrapped the data and include it to the response and returns to the caller. To create resources which will be called by the client side, all resource classes must annotate with @Path annotation at the root level of the resource class. Resource class will be the entry point to the resource service, and all resource services must define what type of data can be consumed and produced and, in this project, as discussed earlier JSON will be the primary data type in all resources. Defining the resource data type is simple, add the @MediaType annotation to the resource root or specific resource method with the value of APPLICATION_JSON.

Booking item resource is the representation of the BookingItemEntity, the REST resource using one booking item service for CRUD operations, the resource service

meant for dealing with customer booking item entity, internally the booking item service uses other services and DAOs for validations and persistence operation.

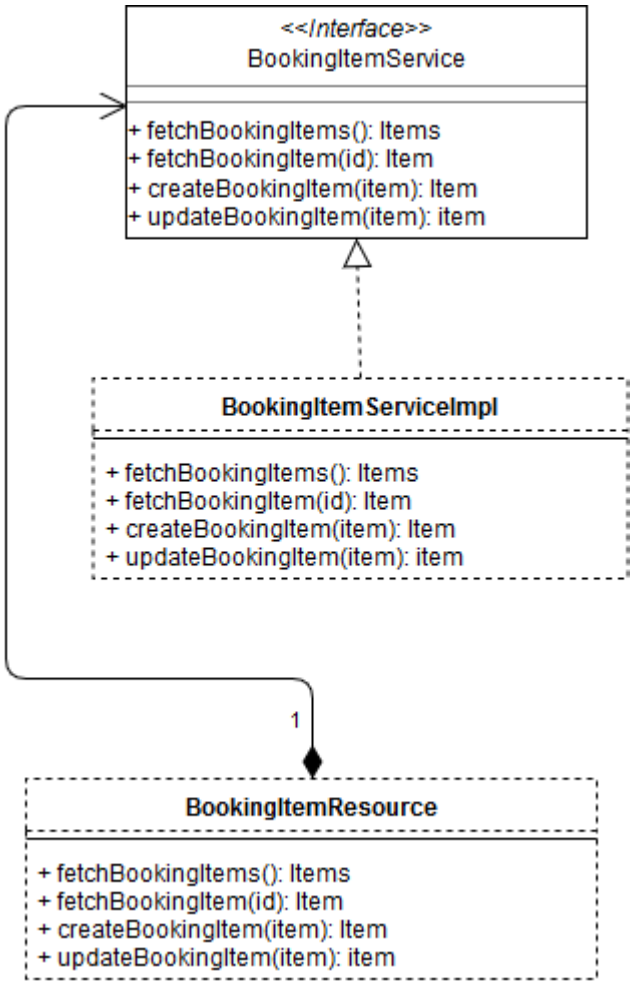


Figure 4. BookingItemResource UML

Figure 4 explain and identify the primary responsibility of the booking item resource. There are four main resource endpoints which meant for fetching all and single booking items, creating new customer booking item and updating customer booking item.

Booking item resource class meant for dealing with linking products to customer booking entity. The resource class uses internally booking item products service as described in figure 5. Figure 5 has three main resource services which used for linking a new product to customer booking item and updating products and deleting them. Booking item products services Internally aggregating other services and validators and DAOs for crud operations.

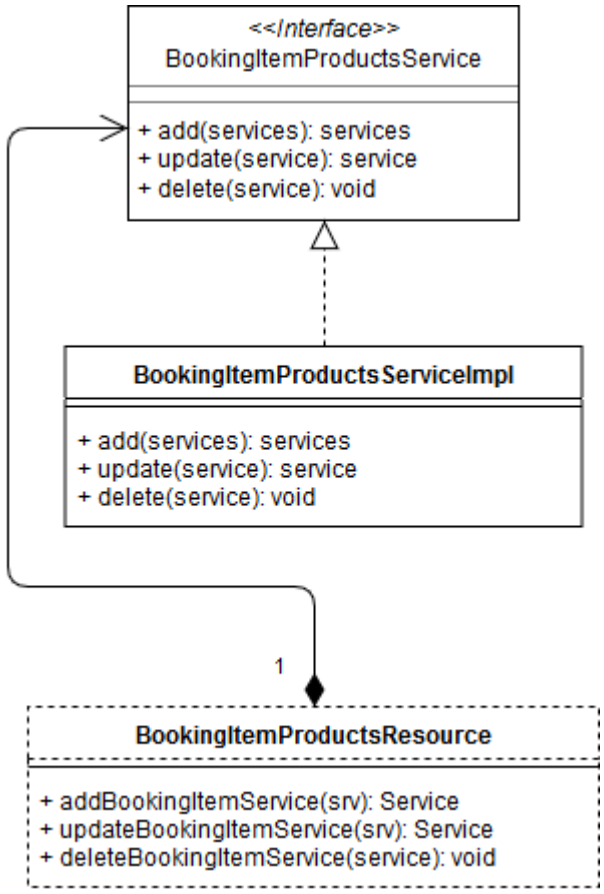


Figure 5. BookingItemProductsResouce UML

Figure 6 discuss the resource class which meant for dealing with customer booking template entity. The booking template resource class uses booking template service as shown in figure 6 which has four primary responsibilities for creating, updating and querying. The booking template services internally aggregate some other services, validators and DAOs for validation and CRUD operations.

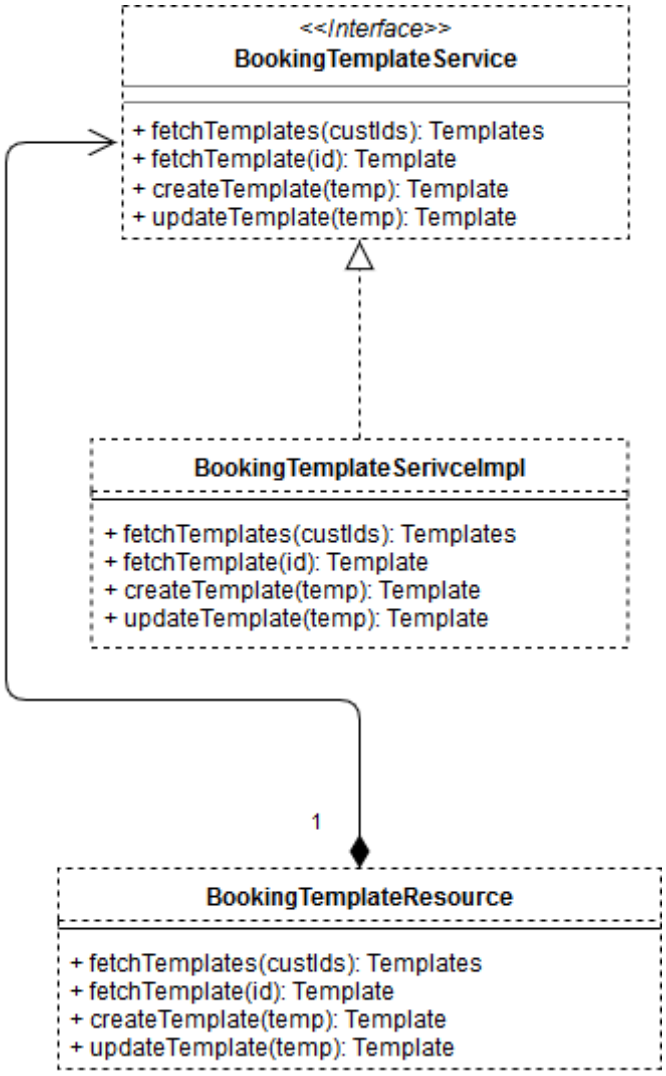


Figure 6. BookingTemplateResource UML.

Figure 7 discuss the resource class which meant for dealing with linking products to customer booking template entity. The resource class uses internally booking template products service as described in figure 7, and it has three mains services which used for linking a new product to customer booking template, deleting and querying them. Booking template products services Internally aggregating other services and validators and DAOs for CRUD operations and validations. From figure 7, note that there is no update API for a specific product because the customer requirement might change daily and does not make sense to have that update for all connected customer booking items. Dealing with template products resource might involve updating all related customer booking items if the booking template has a defined customer repetition. Again, only admins employees can interact with this resource.

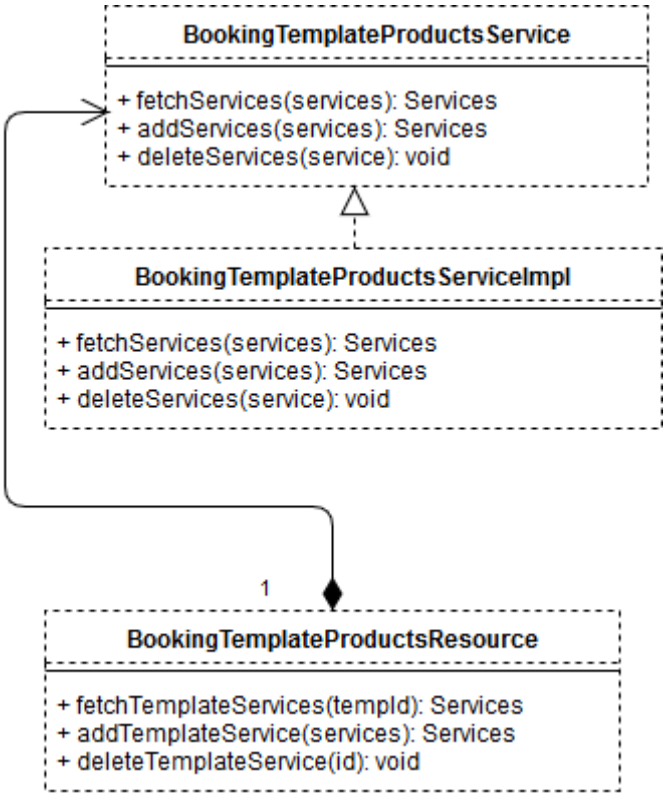


Figure 7. BookingTemplateProductsResource UML.

3.5. Action Diagrams

Activity diagrams are a technique to show procedural logic, business process, and workflow. In many ways, they play a role same to flowcharts, but the principal difference between them and flow chart notation is the support parallel behavior [18]. All the activities explained in this section are describing the workflow and process modeling of each presented use case.

3.5.1. Creating a Customer Booking Item

Create single customer booking item activity as described in the above figure 8, first start by validating the employee privileges. If the employee has the access rights for CRUD setting to the specific customer than validation for the customer item entity payload will be invoked, and here the service will validate all the required attributes otherwise the create operation will fail. If everything okay then persists the payload to database and query available products from booking type and filter out what are products available for the customer, finally store the products to the database and return to the caller.

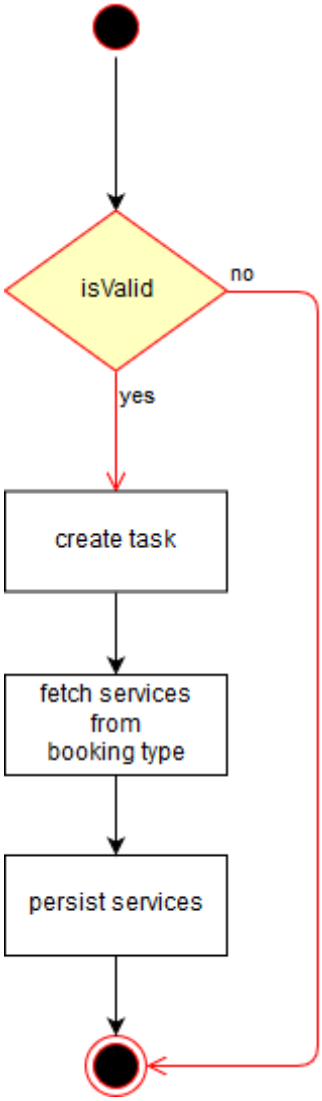


Figure 8. Creating a single customer booking item.

3.5.2. Updating a Customer Booking Item

Update single customer booking item activity as described in the above figure 9, first starting by validating the employee privileges. If the employee has the access rights for CRUD operations to the specific customer than validation for customer item entity payload will be invoked, and here there are a limited number of fields that allowed to be present during the validation. If validation passed then check the current booking item status and if it is finished then end up the operation, if not then the new state needs to be tested. Case item state finished, and then realizing billables is required otherwise update the booking item.

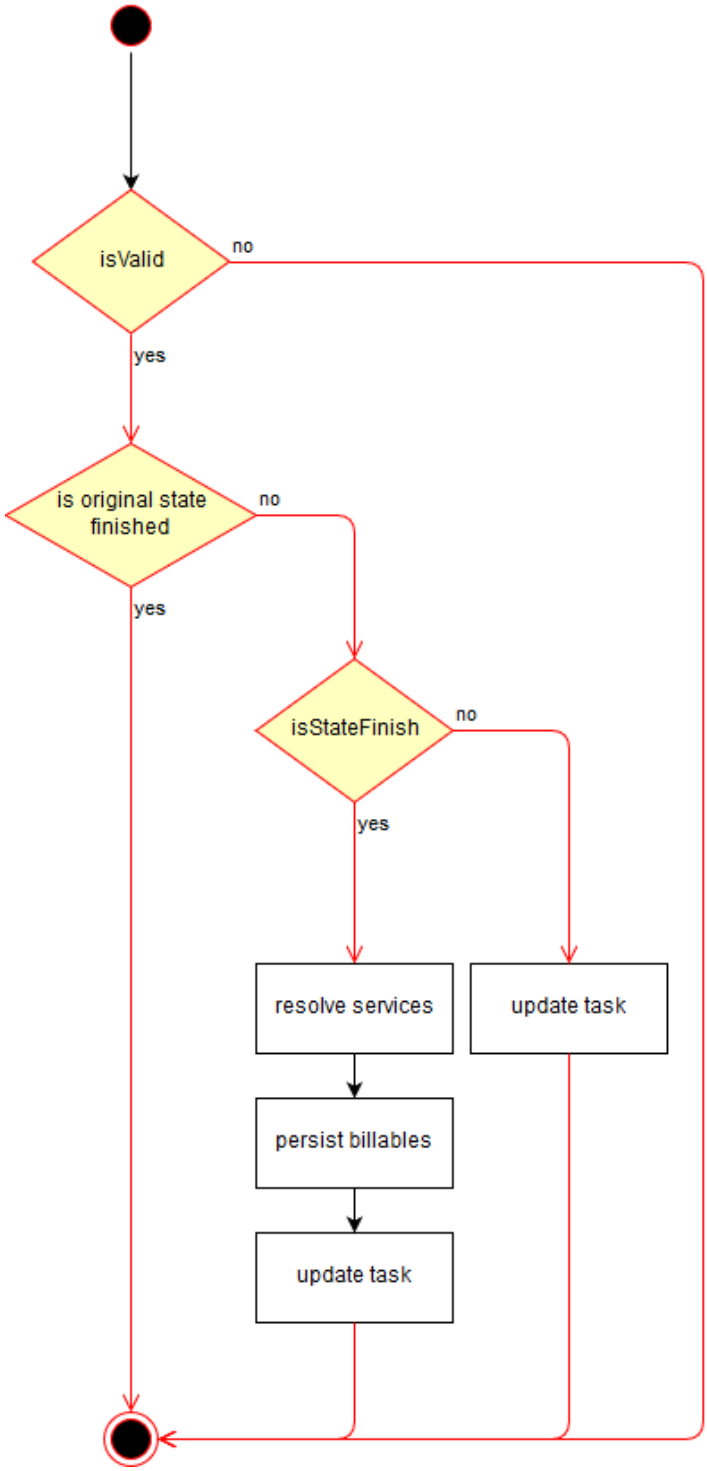


Figure 9. Update a single customer booking item.

3.5.3. Deleting a Customer Booking Item

Delete single customer booking item activity as described in the above figure 10, starting by validating the employee privileges, and if the employee has the access rights for CRUD operations to the specific customer then validation of the existence of the booking item will be invoked and if it exists then check the current status.

Status case finished the end with an error otherwise delete the customer booking item and the attached products.

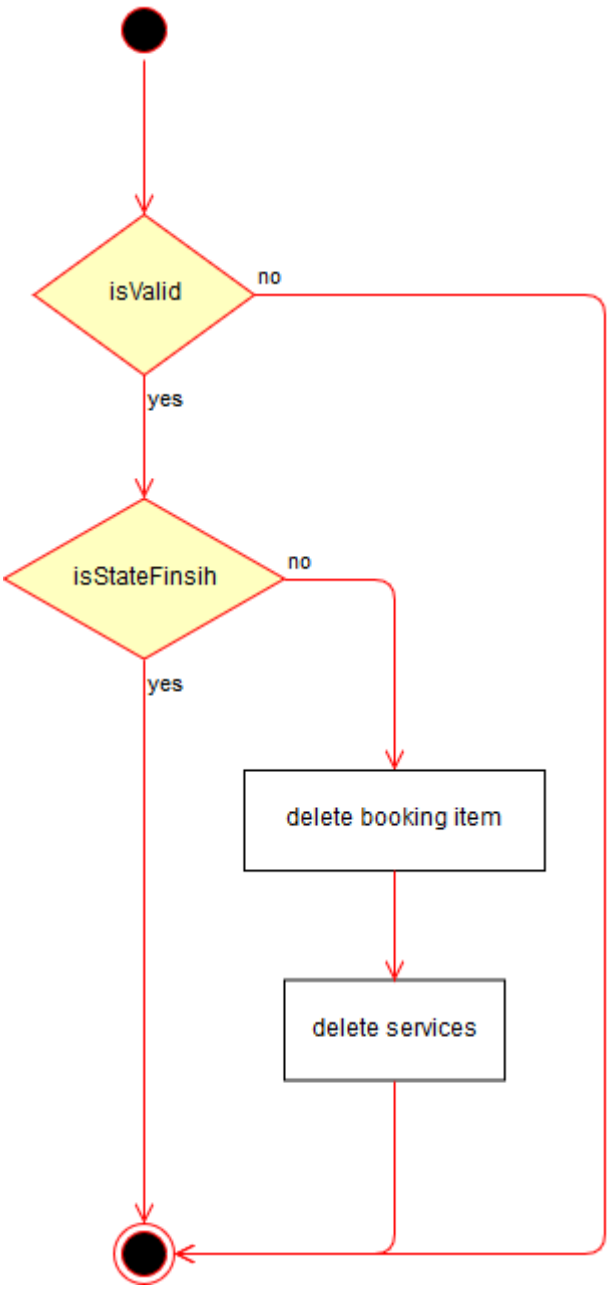


Figure 10. Delete a single customer booking item.

3.5.4. Adding Products

Add products activity as described in the above figure 11, starting by validating the employee privileges, and if the employee has the access rights for CRUD operations to the specific customer entity, then validation of the product payload will be invoked. If things are valid, querying the current customer booking item will be called and validate the current status if it allows adding new products or not, if the state is finished existing with an error otherwise persist new product to the database.

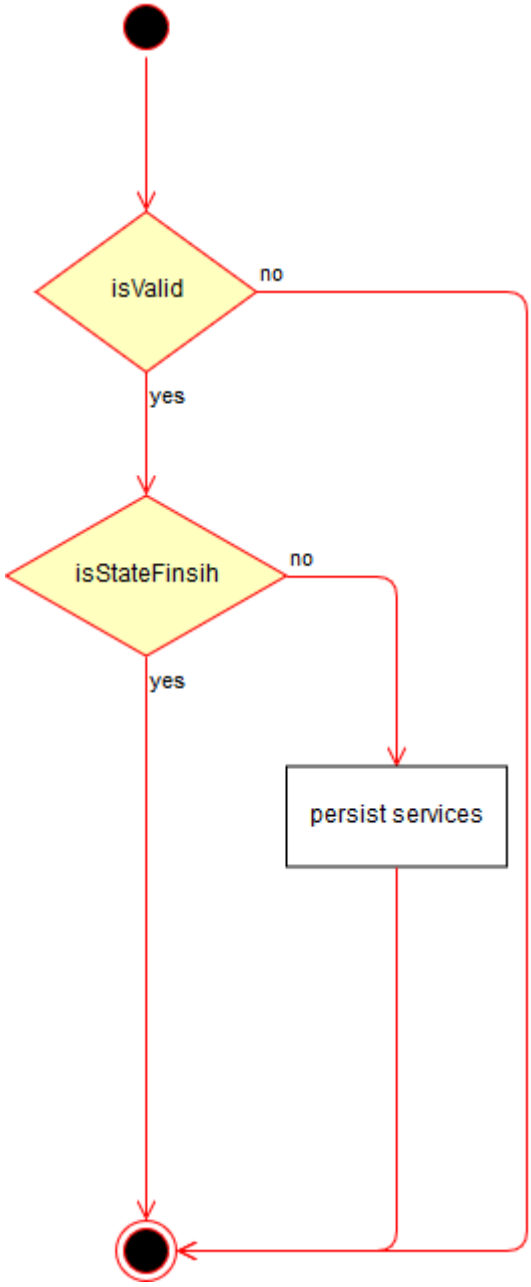


Figure 11. Add services.

3.5.5. Activity update products

Update product activity as described in the above figure 12, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer then confirming the product payload, and it is existence. If everything is valid then querying the current customer booking item and validate the current status. If the current state finished, then exit with an error otherwise update the attached product.

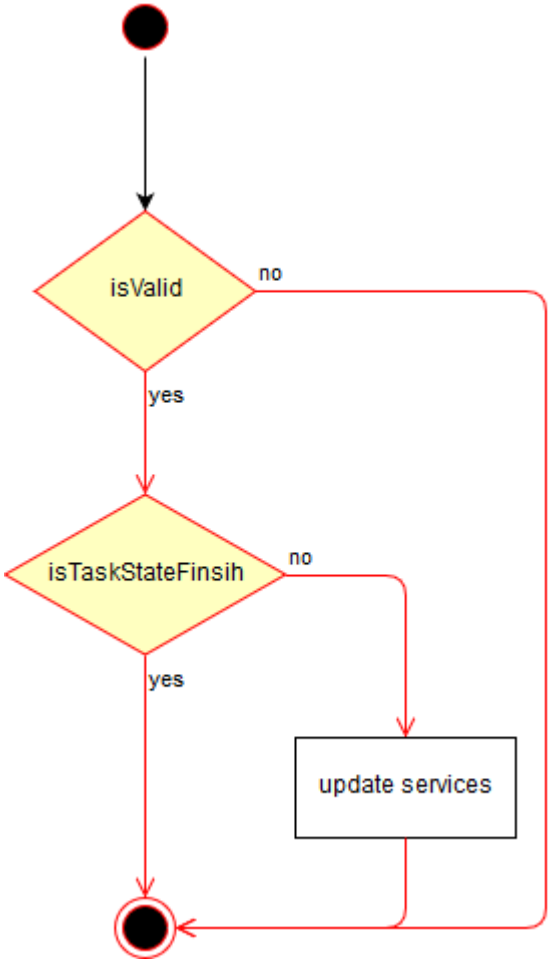


Figure 12. Update services.

3.5.6. Deleting Products

Delete product activity as described in the above figure 13, starting by validating the employee privileges. If the employee has the access rights for CRUD operations to the specific customer, then verify the existence of the product. After things are confirmed, querying the current customer booking item and validate the current status if it finished or not if the case ended then exits with an error otherwise either update the product count or delete the product based on the booking type.

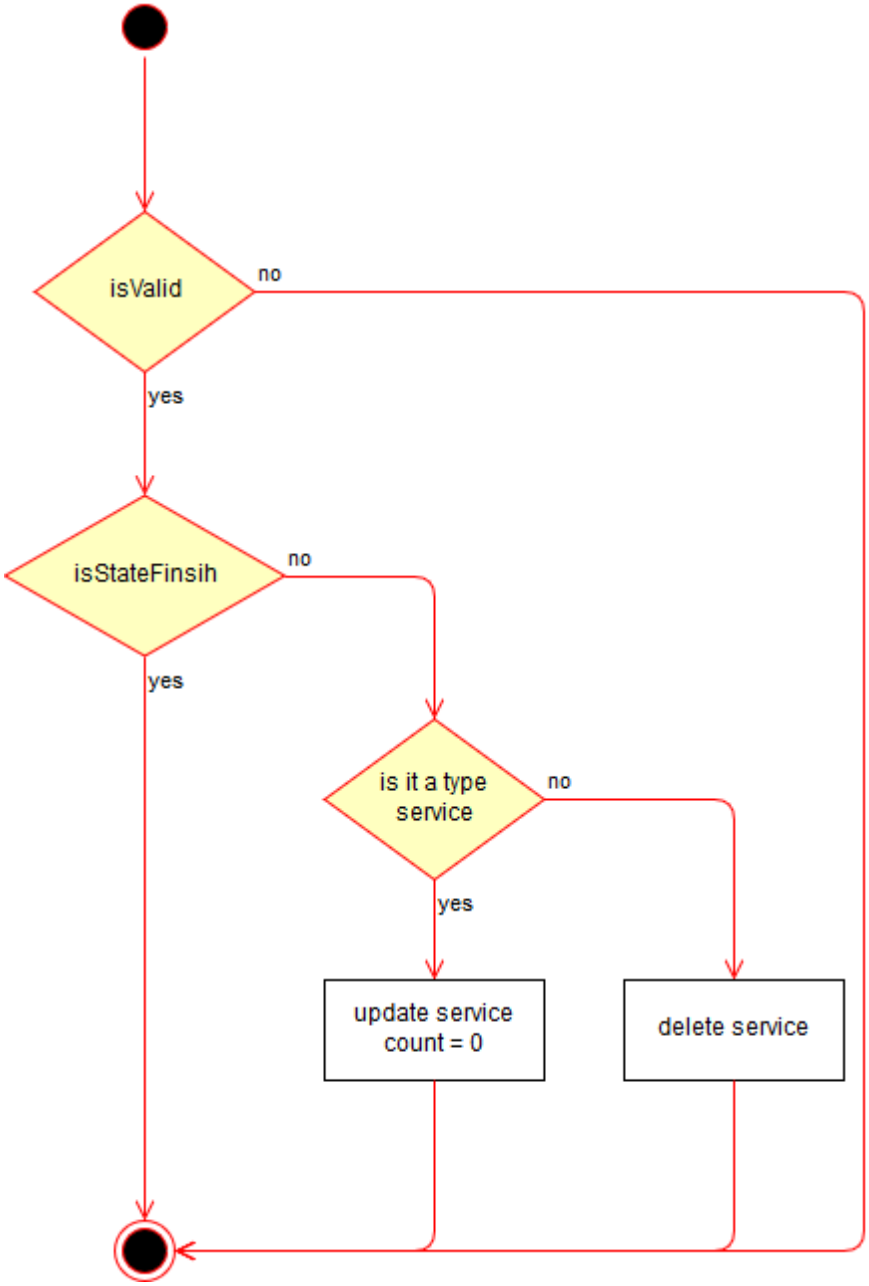


Figure 13. Delete services.

3.5.7. Creating a Customer Template

Create customer booking template activity as described in the above figure 14, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer then validating the customer booking template payload. If validation completed successfully then persist the customer resource template and query the products available to customer resource from the booking type and persist them to the database.

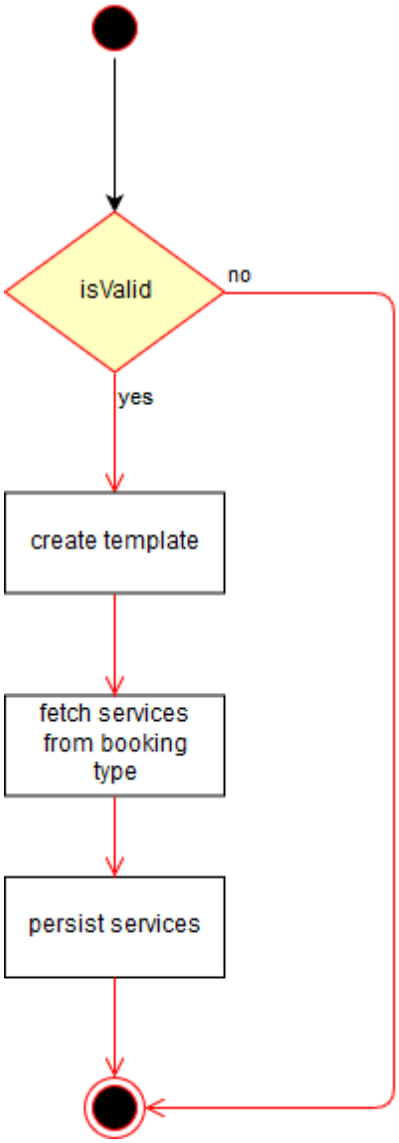


Figure 14. Create a customer booking template.

3.5.8. Updating a Customer Template

Update customer booking template activity as described in the above figure 15, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer then validating the update payload. If validation completed successfully then update the customer template. The template might have a repetition, and if so, then update all connected customer booking items.

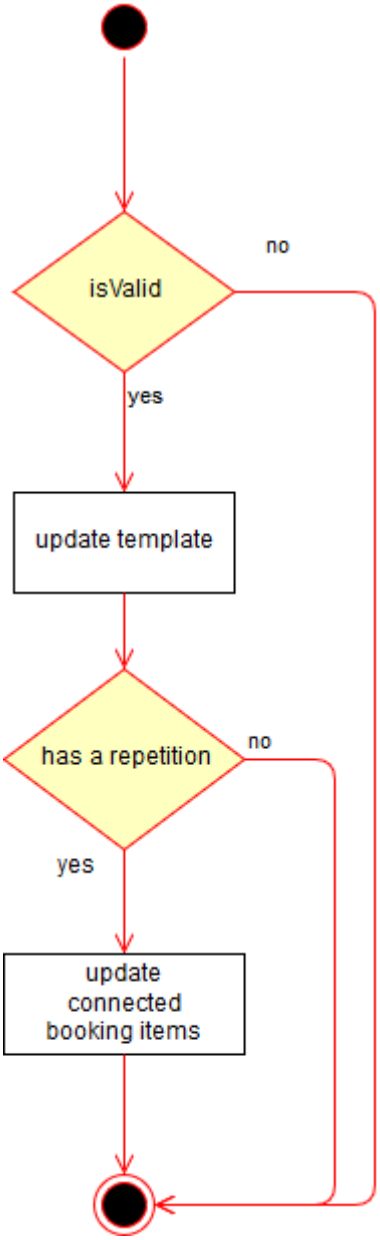


Figure 15. Update customer booking template.

3.5.9. Adding a Template Products

Add products to customer template activity as described in the above figure 16, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer then verifying product payload. If validation completed successfully then persist services. The template may connected to a repetition, query all connected customer booking items and create new products and store them to the database.

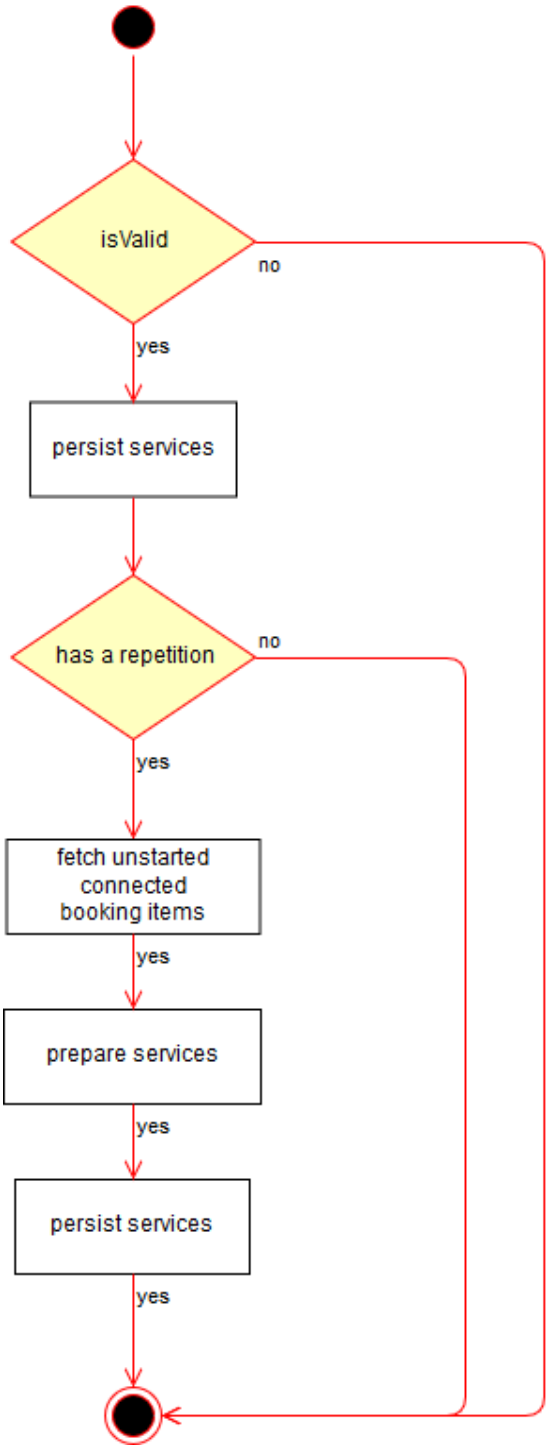


Figure 16. Add template services.

3.5.10. Deleting a Template Products

Delete template service activity as described in the above figure 17, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer then confirming the existence of the product. Then either delete the service or update the service based on the customer

booking type. The template might have to a repetition and in that case, querying all connected booking time and either update products or delete based on customer booking type.

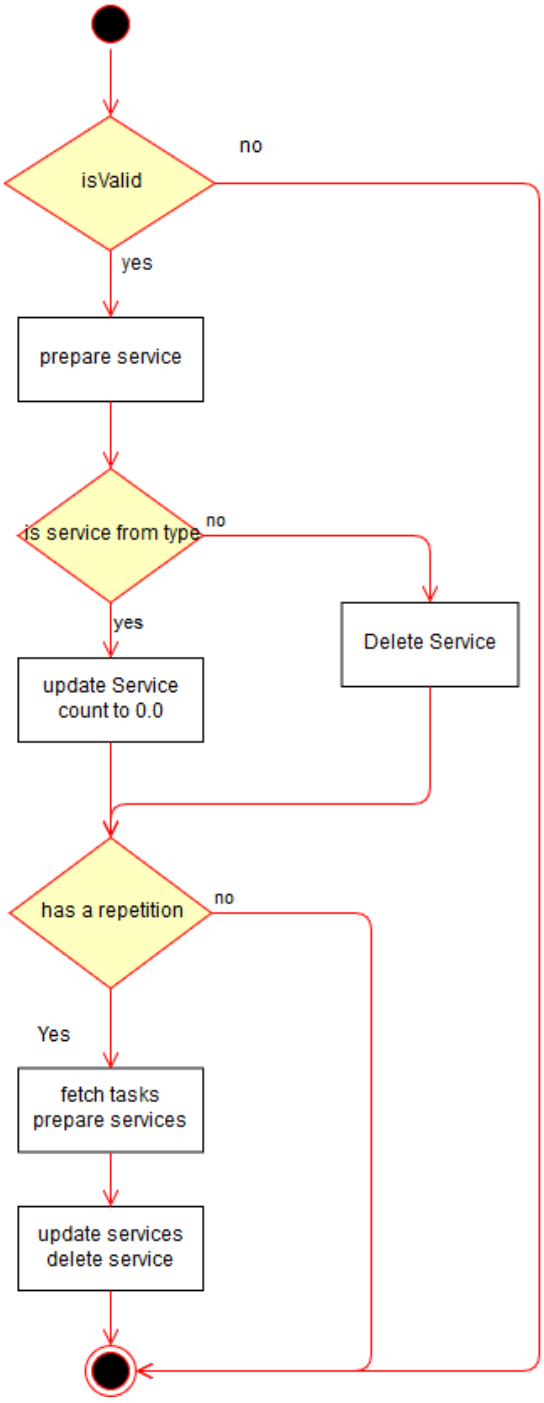


Figure 17. Delete template services.

3.5.11. Creating a Repetition

Create booking repetition activity as described in the above figure 18, starting by validating the employee privileges and if the employee has the access rights for

CRUD operations to the specific customer then confirming the booking repetition entity. After the validation operation completed, persist the customer repetition, the result will be used to generate customer booking items based on the repetition requirement and products to each booking item. Finally, store the created entity items and products.

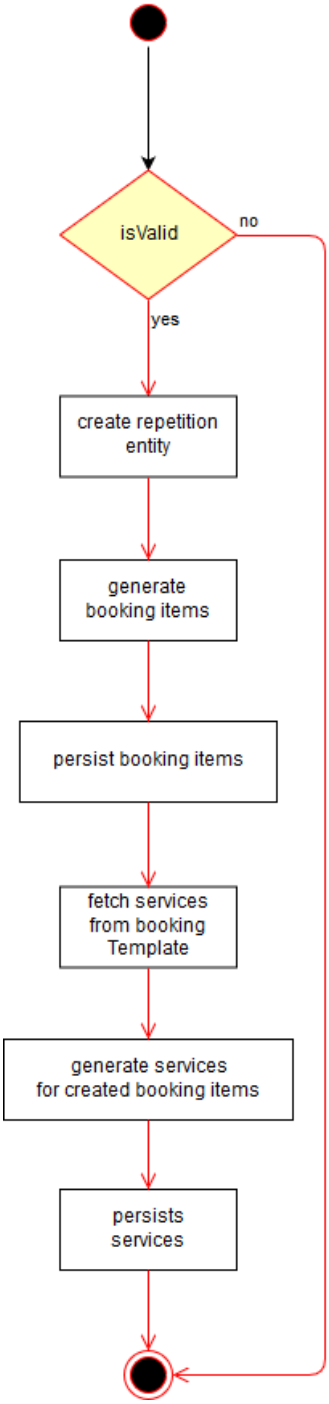


Figure 18. Create booking repetition.

3.5.12. Updating a Repetition

Update booking repetition activity as described in the above figure 19, starting by validating the employee privileges and if the employee has the access rights for CRUD operations to the specific customer and then validating the customer repetition entity. When validation operation completed, generate new booking items based on the provided repetition payload, and query the existing booking items from the old repetition, compare both items and generate a delete and insert operations same goes for products. Finally, persist new booking entity items and products and delete old items and products.

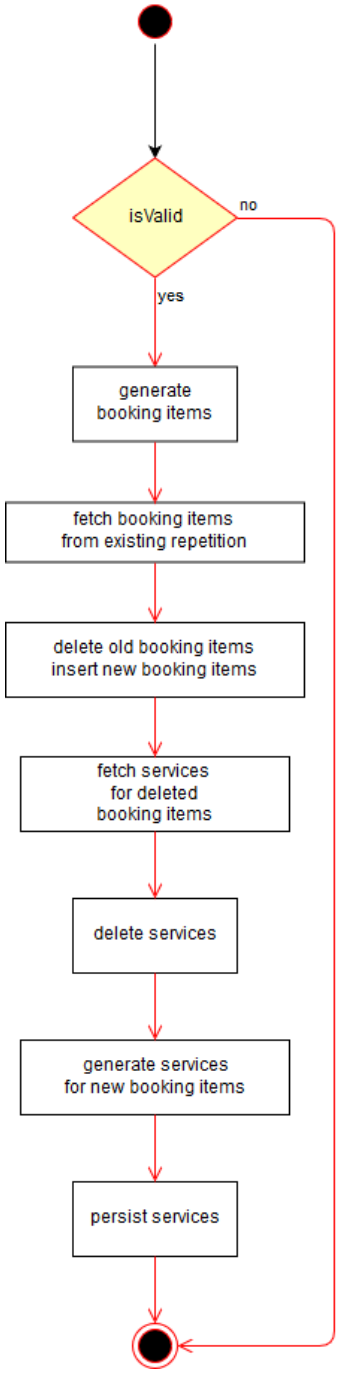


Figure 19. Update booking repetition.

3.6. Sequence Diagrams

For all sequence diagrams, there are always two parties involved in the end to end communication. The employee in practice is the client used by the user which will communicate with the REST APIs.

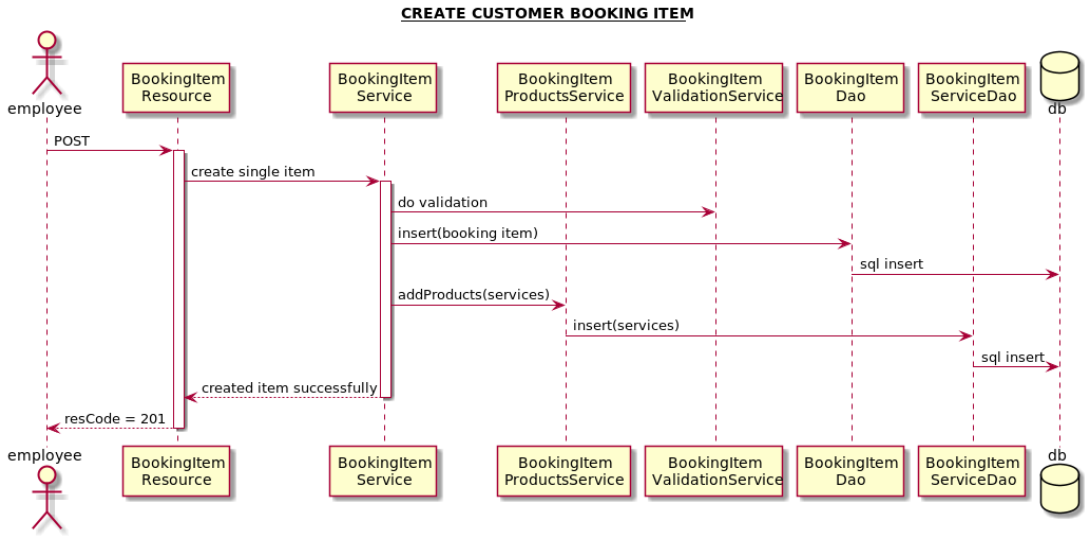


Figure 20. Create booking item.

Figure 20 start by initiating a POST request to booking item resource, the HTTP request included the entity payload. The server will receive the client request process it and direct to the right resource based on URI matching. In the booking item resource, a booking item services will handle the posted payload. Internally in booking service, there are other services, and Dao are involved, every process starts with validating the entity payload and privileges if validation failed an exception will throw with proper error messages. Otherwise, the booking item service will invoke the insert method from booking item Dao. After creating the booking item products will persist if the booking has a defined type. The result from this operation is the created booking item in which will be included in the response body with code 201, which means that item created.

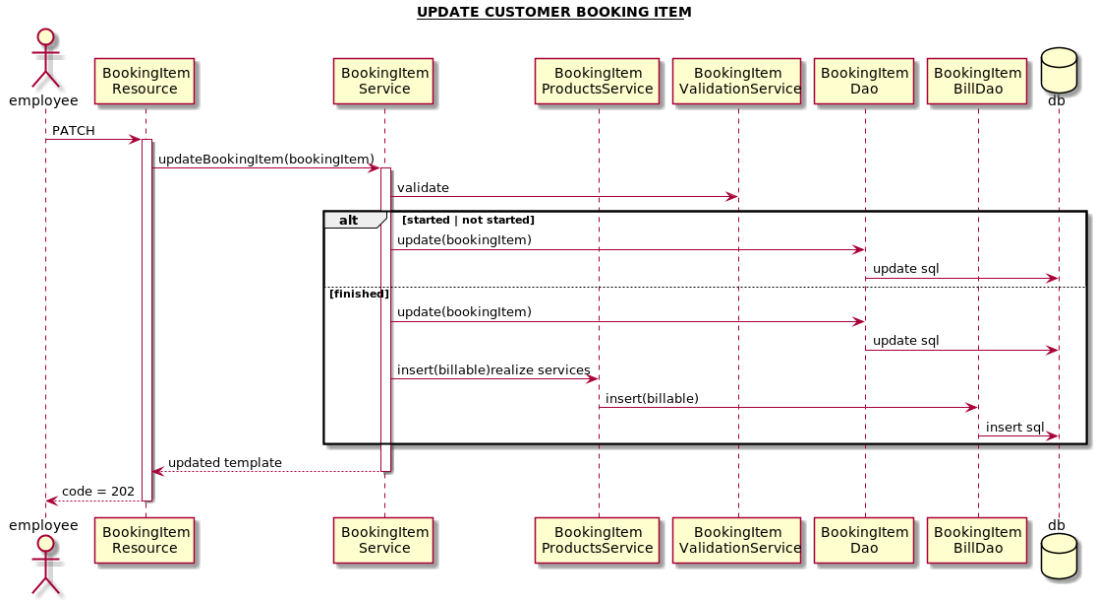


Figure 21. Update booking item.

Figure 21 Update booking item sequence start by initiating a PATCH request to booking item resource with the booking item entity payload. Booking item service will invoke update function, which will handle the update operation. As discussed, above the booking item service internally will use some other services and DAOs. After validating the payload, there are two cases based on the new booking status. Bookings status are (FINISHED, STARTED, NOT_STARTED) if the new state is finished then update the entity thru the booking item DAO and realize the attached products and create billables, otherwise a regular update to the resource entity without realizing any products.

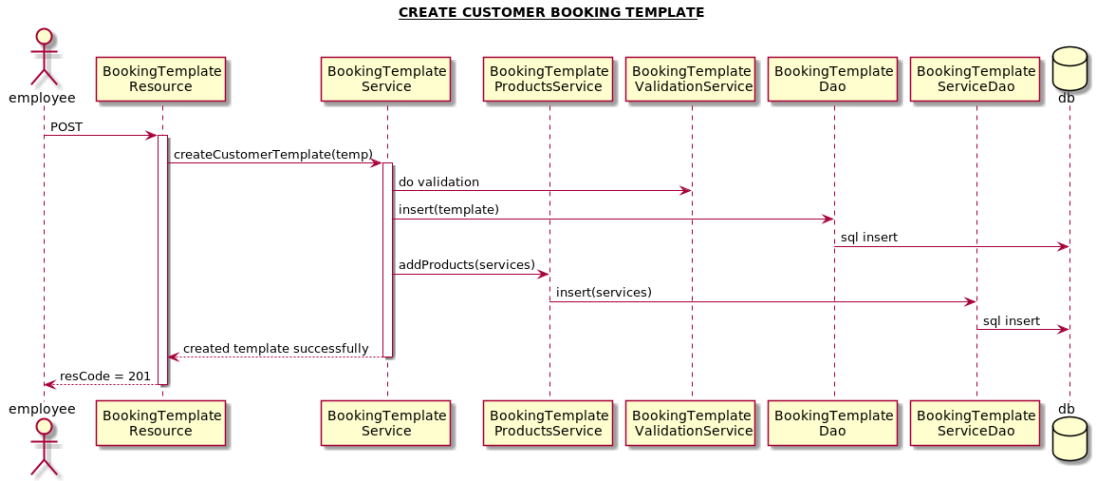


Figure 22. Create a customer template resource.

Figure 22 create customer resource template is very similar to creating a booking item except that the model does not have a status and employee attributes.

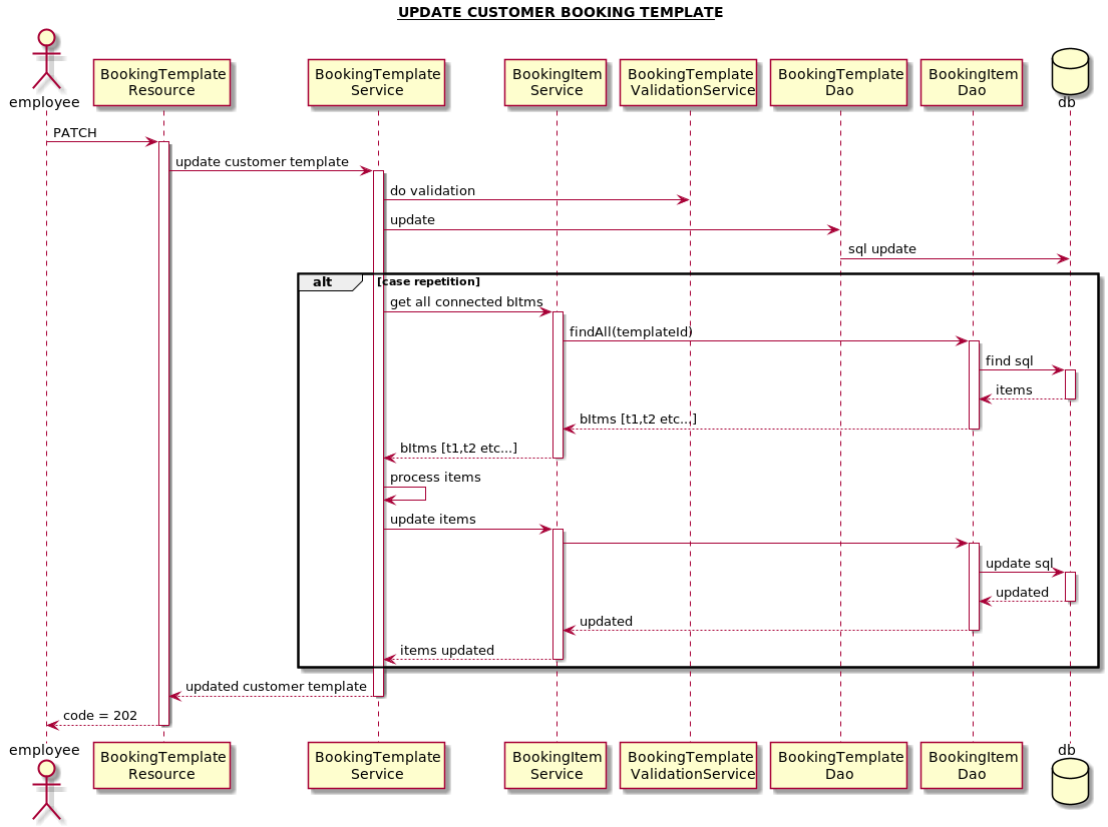


Figure 23. Update a customer template resource.

Figure 23 Update booking template is very similar to updating customer booking item except that if the customer template has connected repeated booking items, the update will take effect all connected non-started booking entity items.

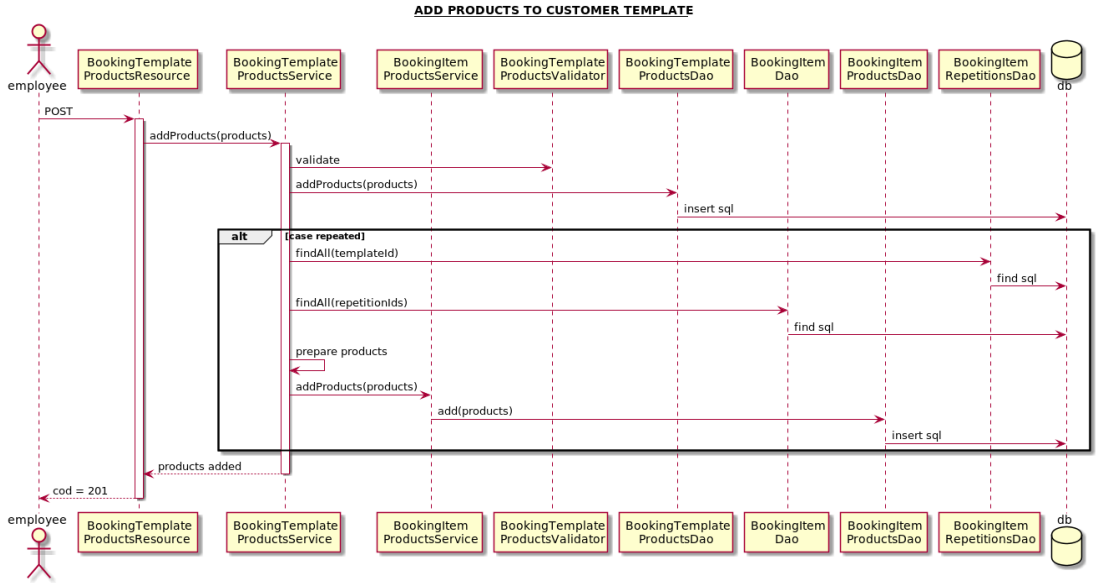


Figure 24. Add products to a template.

Figure 24 Adding products to customer template share the same functionality comparing to booking item but besides, if the model has a connected repetition then all related booking item will contain the newly added products.

Figure 25 Similarly to delete products from the template is done the same way except that if there is repetition with connected booking items the delete operation will take effect as well to all connected ones.

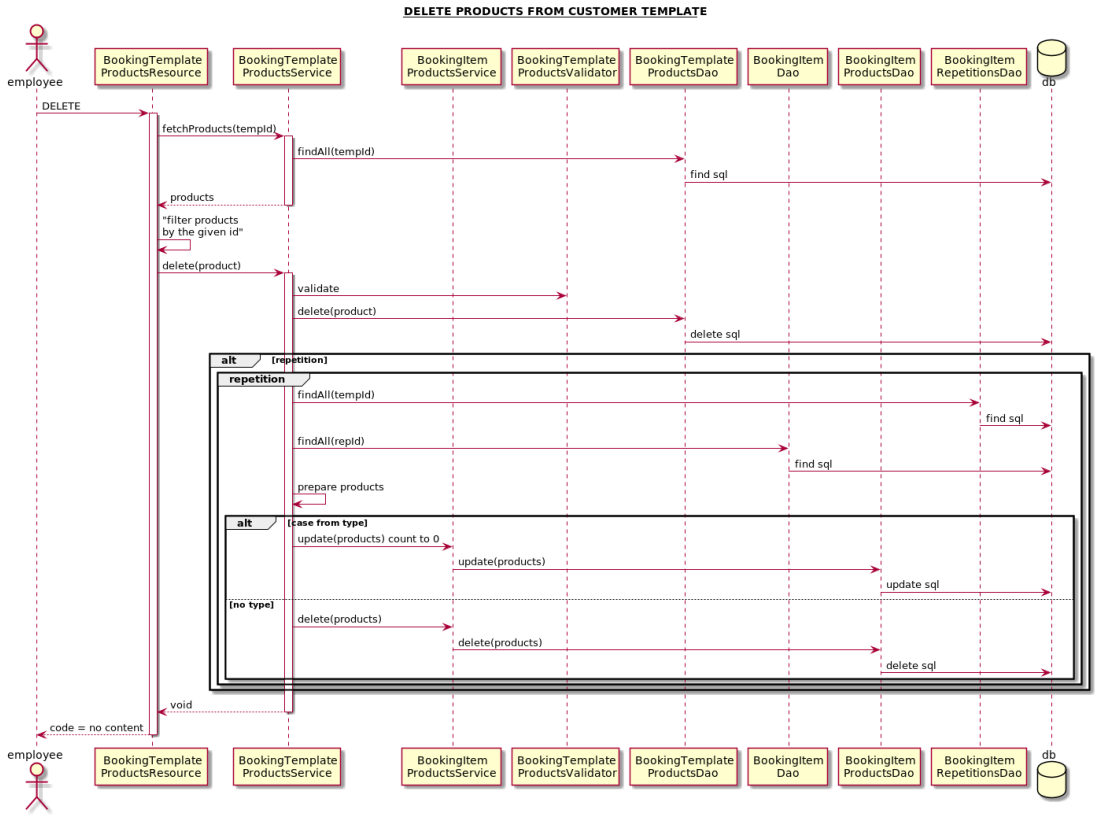


Figure 25. delete product from template.

Figure 26 Adding products to customer booking item start by initiating POST request to booking item products resource with product payload. Products service will handle the processing of the HTTP request, and internally the product service uses other services and DAOs for validation and persistence operations. The result is the created entities included in the response with the status code of 201.

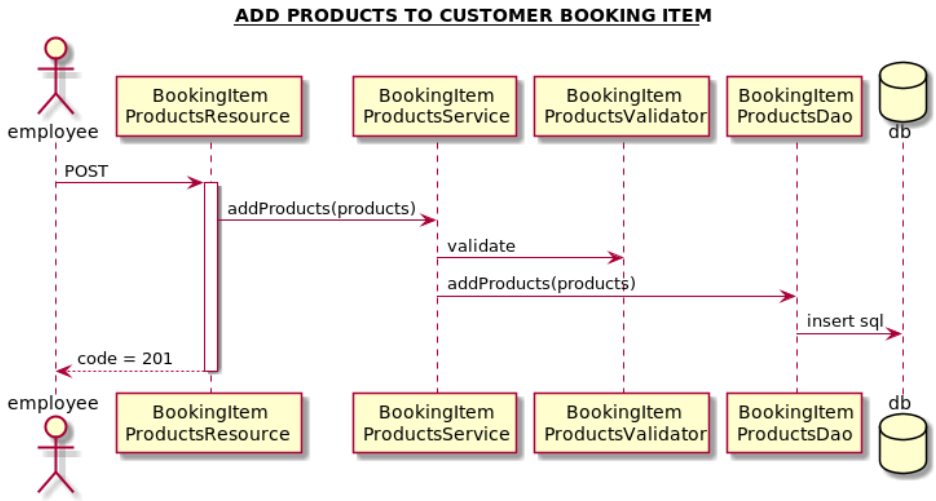


Figure 26. Add products to a booking item.

Figure 27 Updating customer booking item products by the client start by initiating PATCH request to booking item product resource. Product service will handle the request by first validating the entity payload then uses the product DAO for merging the changes to the database. There is no update API for customer template products as discussed earlier above in the activity diagrams.

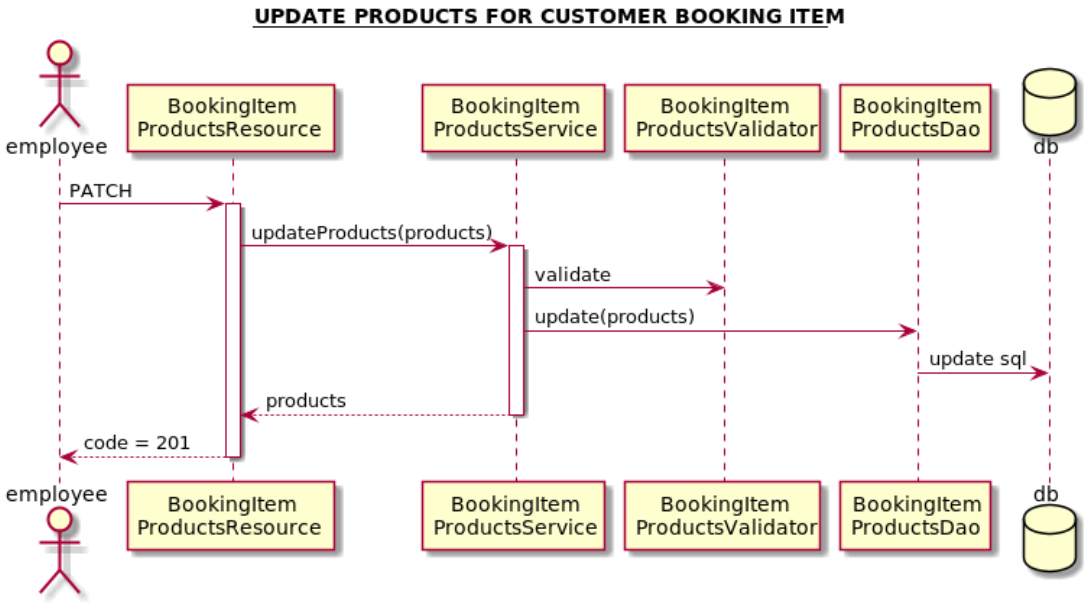


Figure 27. Update products to a booking item.

Figure 28 Deleting products from booking item start by initiating DELETE HTTP request by providing the resource to be deleted in the URI path param. The products service will handle the request and internally do the validation and remove operation.

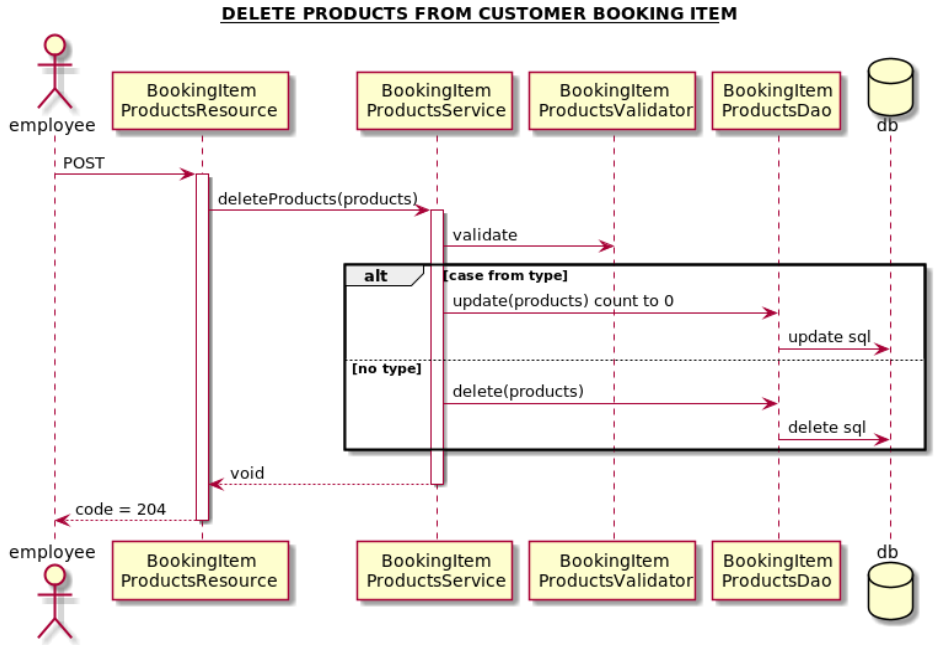


Figure 28. Delete products from a booking item.

Figure 29 Create repetition is starting by initiating HTTP POST request, the booking repetition service will handle the processing of the HTTP request, first by validating it and if validation approved then persist the repetition entity to the database.

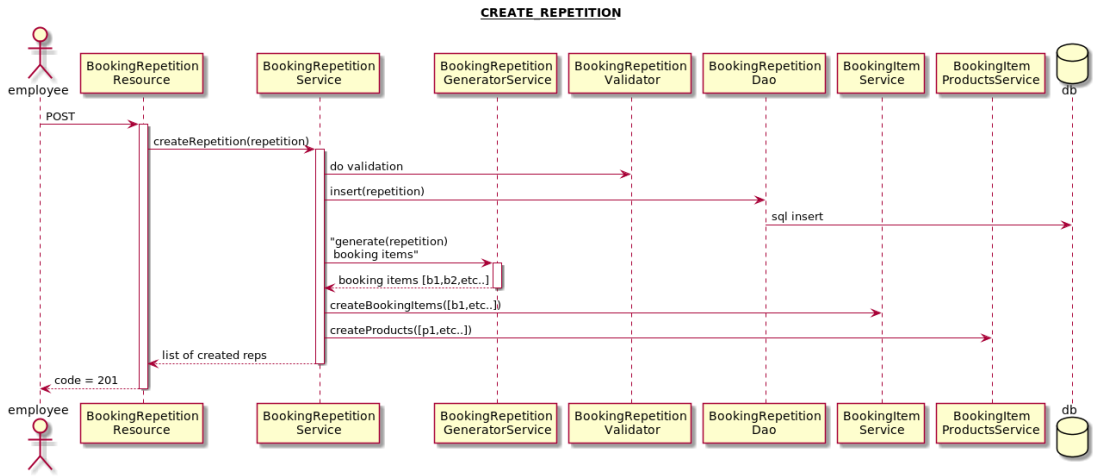


Figure 29. Create repetition.

Generating booking items from the persisted entity will result is a collection of booking items entities which are ready for persistence, once the booking items entities stored to the database, generating products must do. The service will query all attached products belonging to a booking entity and then generate new products for all created booking items. Finally persisting the products and return the created repetition entity including it in the response.

3.7. Services and DAOs

The services package contains all classes, which part of the domain layer. The services classes as shown in UML diagrams and sequence diagrams are defining the business logic. Internally the services will be responsible for validating the employee privileges and validating the entity based on specified constraints. The services will communicate back and forth with the data source layer for persistence operation, mainly the communication will be done through the data access objects DAOs. DAOs classes main responsibility is to interact with the database and all CRUD operations. The JOOQ is used as main API framework as discussed earlier in the environment section. The project communicates with one primary database, which is a MySQL database server.

4. TESTING AND VALIDATION

After the API implementation, unit tests are necessary. Tests created under the test package; units test is crucial in every project as they are making sure all components working properly. All unit tests in this thesis are functional tests, and there is another type of experiments conducted in this thesis other than the functional unit tests such as packaging the application and deploying the packaged artifact to the tomcat server.

This section presents both functional unit tests and server environment test. Functional unit tests are implemented using jersey and Junit test frameworks. Jersey framework is compelling and easy to use, the jersey will spin up for every unit test an in-memory tomcat container for testing the API, the JUnit framework is compelling for creating unit tests, JUnit APIs have all kind of useful function for assertion the data obtained for the unit tests. Each test method directly annotated with @Test annotation, the test method must be declared as public, and the return type must be void.

4.1. Testing

There are two primary use cases for units testing, and the first use case is to test an employee with admin access privilege, second is an employee with fewer rights than admin. Admin can read, create, update and delete customer templates and customer repetitions. A regular employee can read, create, update and delete customer-booking item. APIs, which used for functional unit tests are.

- GET /booking/bookingtemplates
- GET /booking/bookingtemplates/{id}
- POST /booking/bookingtemplates
- PATCH /booking/bookingtemplates/{id}
- POST /booking/bookingtempaltes/{id}/services
- DELETE /booking/bookingtempaltes/{id}/services/{serviceId}
- GET /booking/bookingitems
- GET /booking/bookingitems/{id}
- POST /booking/bookingitems
- PATCH /booking/bookingitems/{id}
- POST /booking/bookingitems/{id}/services
- PATCH /booking/bookingitems/{id}/services/{serviceId}
- DELETE /booking/bookingitems/{id}/services/{serviceId}
- POST /booking/bookingrepetitions
- PATCH /booking/bookingrepetitions/{id}

For the user to able to access the REST APIs, the user must be logged in to the system. Once the user logged in, the server will set the cookies header to the user with the correct role based on the user credentials. The login process is not part of this thesis. All APIs has a proper exception handling with a clear error code and human-readable error message. All validation services throwing an appropriate exception with a predefined error code and error message. The assert Equals method, one of the powerful JUnit methods which will be used intensively for all unit tests.

There are a couple of test cases, testing the API response code is one of the most common in this thesis. Response code tells if the HTTP request successfully processed and handled back to the client, if not the assert equals will fail with printing out the exception message with the API response code. All APIs returns standards HTTP response code, and codes used as below.

- 200 OK
- 201 CREATED
- 202 ACCEPTED
- 204 NO CONTENT
- 400 BAD REQUEST
- 403 FORBIDDEN
- 401 UNAUTHORIZED

The response will contain the results in the body, and deserialized from JSON object to java object in the client side.

4.1.1. Creating a Customer Template Test

Creating a customer template entity can be only done by a user with admin privileges, to create the template, the POST API for customer template entity must be called with template payload as demonstrated in figure 30.


```
{
    "startTime": "2019-01-01T16:00:00",
    "EndTime": "2019-01-04T17:00:00",
    "desc": "creating customer template test",
    "customerId": 1,
    "itemTypeId": 1
}
```


Figure 30. Customer template payload.

All attributes of the template entities must exist. Otherwise, the requested API will fail in the validation stage with response code 400. If the user trying to access this API is not a privileged user, a 403 will return to the client side.


```
{   "id":1,
    "startTime": "2019-01-04T16:00:00",
    "EndTime": "2019-01-04T17:00:00",
    "desc": "creating customer template test",
    "customerId": 1,
    "itemTypeId": 1
}
```


Figure 31. Template representation.

4.1.2. Updating a Customer Template Test.

There are few attributes allowed to modified for the customer template, and all updatable attributes must provide in the payload. In this test, the only description is included and updated., an example of the payload is in figure 32.

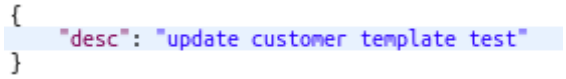


Figure 32. Template update payload.

If the update request succeeded, the server returns an HTTP response with status code 202. The response body includes the updated resource as in figure 33.

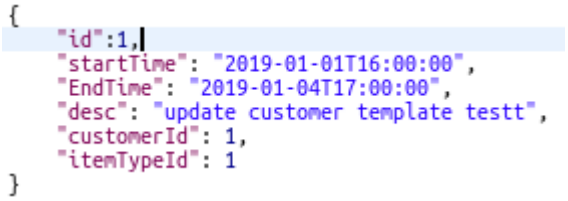


Figure 33. Template update representation.

In case the payload failed during validation layer, the API will return status code 400 which is an HTTP bad request with a proper error message. If the template has a connected repetition, the update will affect all related booking items.

4.1.3. Adding Products to Template Test

Template without products is useless. Products entities are the billable services for a customer after realizing some tasks. In this test, three services added to customer template which will be used later in the generated booking items. Services payload is in figure 34.

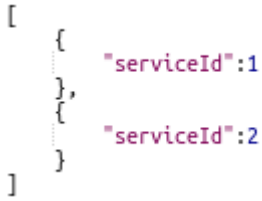


Figure 34. Products add payload.

After request completed, API return code 201 which created and, in the HTTP response, the generated products resources will be returned. The response representation is in figure 35.

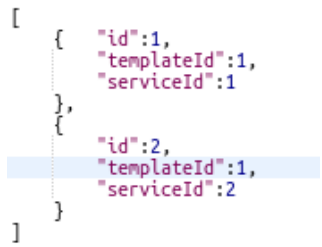


Figure 35. Products add representation.

HTTP Request can fail during the validation stage for the case where adding a product which does not exist or not allowed for the targeted customer.

4.1.4. Deleting Products from Template Test

Deleting service API can be invoked by providing the service id to be removed if the request succeeded, no content response with status code 204 otherwise bad request with status code 400 returned.

4.1.5. Planning Tasks in Future Test

To plan customer booking item entities, it must create a template entity at first. Once customer template is created, invoking this API required to defend the type of repetition with start date and end date. Repetition payload is in figure 36.

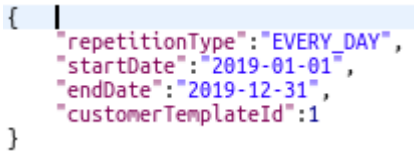


Figure 36. Repetition payload.

Once request succeeded, response with status code 201 with the created resource returned. An example of an HTTP response is in figure 37.

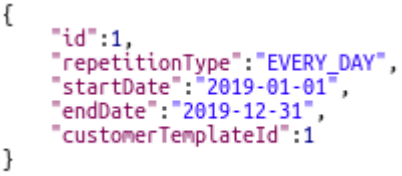


Figure 37. Repetition representation.

Validation will assert all required attributes any missing attribute will cause the validation to fail, and status code 400 returned.

4.1.6. Updating Planned Repetition Test

There is only one field which can be modified which is the end date, in this test, the end date set to 28 February 2020, this will result in the current customer repetition to extended to two months. Repetition update payload is in figure 38.

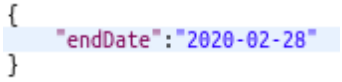


Figure 38. Repetition update payload.

If the customer requirement changed, repetition could be reduced and ended as well. Setting up the repetition end date to the start date of the repetition or current time will result in ending the repetition. Repetition representation is in figure 39.

```
{
    "id":1,
    "repetitionType":"EVERY_DAY",
    "startDate":"2019-01-01",
    "endDate":"2020-02-28",
    "customerTemplateId":1
}
```


Figure 39. Repetition update representation response.

4.1.7. Creating Customer Tasks Test

Fetching the created customer booking items entities is trivial. Query param must exist in the request. Repetition Id param will set to the HTTP request, if the request succeeded, HTTP response with status code 200 returned including the queried resources, the result is in Figure 40.


```
[
    {   "id":1,
        "startTime": "2018-01-01T16:00:00",
        "EndTime": "2018-01-01T17:00:00",
        "desc": "creating customer booking item",
        "customerId": 1,
        "itemTypeId": 1,
        "repetitionId":1
    },
    {   "id":2,
        "startTime": "2018-01-02T16:00:00",
        "EndTime": "2019-01-02T17:00:00",
        "desc": "creating customer booking item",
        "customerId": 1,
        "itemTypeId": 1,
        "repetitionId":1
    },
    {   "id":3,
        "startTime": "2018-01-03T16:00:00",
        "EndTime": "2019-01-03T17:00:00",
        "desc": "creating customer booking item",
        "customerId": 1,
        "itemTypeId": 1,
        "repetitionId":1
    }
]
```


Figure 40. List of booking items representation response.

4.1.8. Creating Task Test

A user with less privilege can create individual tasks and assign them directly to them self without the need for template and repetition. This use case needed for scenarios where the customer requirement has changed, and additional tasks required to fulfill the customer needs. The payload looks similar to the one for the customer template except that it has the employeeId set. Example of the payload is in figure 41.


```
{
    "startTime": "2019-01-01T16:00:00",
    "EndTime": "2019-01-01T17:00:00",
    "desc": "creating customer booking item",
    "customerId": 1,
    "itemTypeId": 1,
    "employeeId":12
}
```


Figure 41. Booking item create entity payload

If the HTTP request succeeded, HTTP response with status code 201 returned including the created resource in the response body. Response representation is in figure 42.

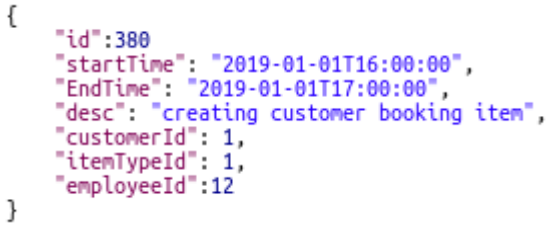


Figure 42. Booking item creates a representation response.

Any missing attributes can fail the request in the validation stage and response with status code 400 will be returned.

4.1.9. Updating Task Test

There are few attributes allowed to be modified for booking item but for the regular user only item type and description are allowed. Example of update payload is in the following figure.

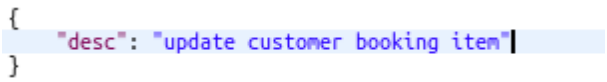


Figure 43. Booking item update payload.

If the HTTP request succeeded, HTTP response code with 202 returned including the updated resource in the response body. Example of response representation is in figure 44. The HTTP response can fail in the validation stage, HTTP response with status code 400 will return.

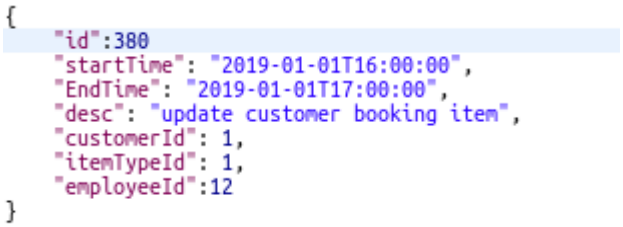


Figure 44. Booking item update representation response.

4.1.10. Deleting Customer Task Test

The user can delete booking item entity if the user has the delete privilege and the owner of the task or the user is the admin. Invoking this API with providing the booking item id in the path param will result in deleting the resource. The response returned is no content with code 204. If the item resource not found, HTTP response code will be 400 with a proper error message.

4.1.11. Reading Customer Tasks Test

Users can list all booking items which are assigned to them, invoking this API for the user with id 12 will query all items resources which assigned to this user. Example of the result is in the following figure.


```
[
    {
        "id":380
        "startTime": "2019-01-01T16:00:00",
        "EndTime": "2019-01-01T17:00:00",
        "desc": "update customer booking item",
        "customerId": 1,
        "itemTypeId": 1,
        "employeeId":12
    },
    {
        "id":381
        "startTime": "2019-01-01T16:00:00",
        "EndTime": "2019-01-01T17:00:00",
        "desc": "create customer booking item",
        "customerId": 2,
        "itemTypeId": 5,
        "employeeId":12
    },
    {
        "id":382
        "startTime": "2019-01-01T16:00:00",
        "EndTime": "2019-01-01T17:00:00",
        "desc": "create customer booking item",
        "customerId": 10,
        "itemTypeId": 2,
        "employeeId":12
    }
]
```


Figure 45. Booking item list representation response.

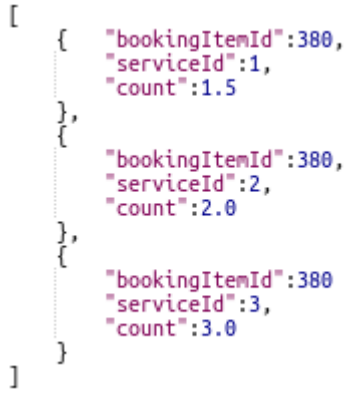
4.1.12. Adding Products to Task Test

Users can add products to their assigned booking items. Linking additional products to specific booking items resources depend heavily on customer needs. As discussed earlier that customer needs are changing, adding products on the fly is convenient. Example of products payload is in the following figure.


```
[
    {
        "serviceId":1,
        "count":1.5
    },
    {
        "serviceId":2,
        "count":2.0
    },
    {
        "serviceId":3,
        "count":3.0
    }
]
```


Figure 46. Products add payload.

If the HTTP request succeeded, the API returns all created resources with response code 201. Example of result representation is in the following figure.



```
[
    {   "bookingItemId":380,
        "serviceId":1,
        "count":1.5
    },
    {
        "bookingItemId":380,
        "serviceId":2,
        "count":2.0
    },
    {
        "bookingItemId":380
        "serviceId":3,
        "count":3.0
    }
]
```


Figure 47. Products add representation response.

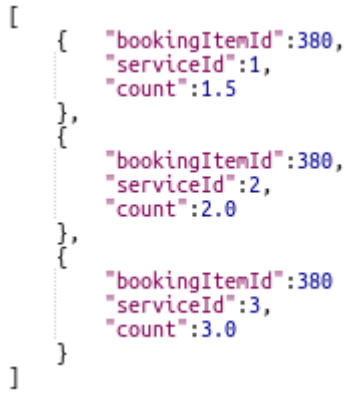
4.1.13. Updating Task Products Test

The user can update products to specific booking item, the case where updating products needed is when the customer asks for the same service many times. Example of such service is giving customer bread, and default could be one, but the customer can request more. Example of update products payload is in the following figure.

```
{
    "serviceId":1,
    "count":3.5
}
```


Figure 48. Products update payload.

Result from the invoked API if succeeded will contain the updated resource with HTTP response code 202. Example of the result is in the following figure.



```
[
    {   "bookingItemId":380,
        "serviceId":1,
        "count":1.5
    },
    {
        "bookingItemId":380,
        "serviceId":2,
        "count":2.0
    },
    {
        "bookingItemId":380
        "serviceId":3,
        "count":3.0
    }
]
```


Figure 49. Products update representation.

4.1.14. Deleting Task Products Test

Deleting service API can be invoked by providing the service id to be removed if the HTTP request succeeded, no content HTTP response with status code 204 returned

otherwise bad HTTP request with status code 400 returned. The user can remove service from booking item if the booking item status not done.

4.2. Validation

The system was designed and implemented based on the information obtained in chapter two. The constraints discussed earlier in chapter two, and the REST API was implemented to be stateless based on section 2.4.2 which add restrictions to the server side so that holding any client info or session is not allowed but stored at the client side. Moreover, based on section 2.4.1, the Client-server components have existed during the development, and Jersey client was used to issuing requests to the server component and receive the response from the server component, the server component used was Tomcat. Also, based on section 2.4.4, in this thesis the communication between the client-server components is done through uniform interfaces, notice that all implemented APIs has well-defined uniform interfaces. Based on section 2.4.3, all API theoretically cacheable based on statelessness but in this thesis cache policy was not part of the implementation. All implemented APIs communicate via HTTP, where HTTP is stateless application-level protocol based on section 2.6 and 2.7, and in this thesis, standard HTTP methods were used based on section 2.7. There are three different validation environments which are development, quality assurance, and release candidate as described in figure 50.

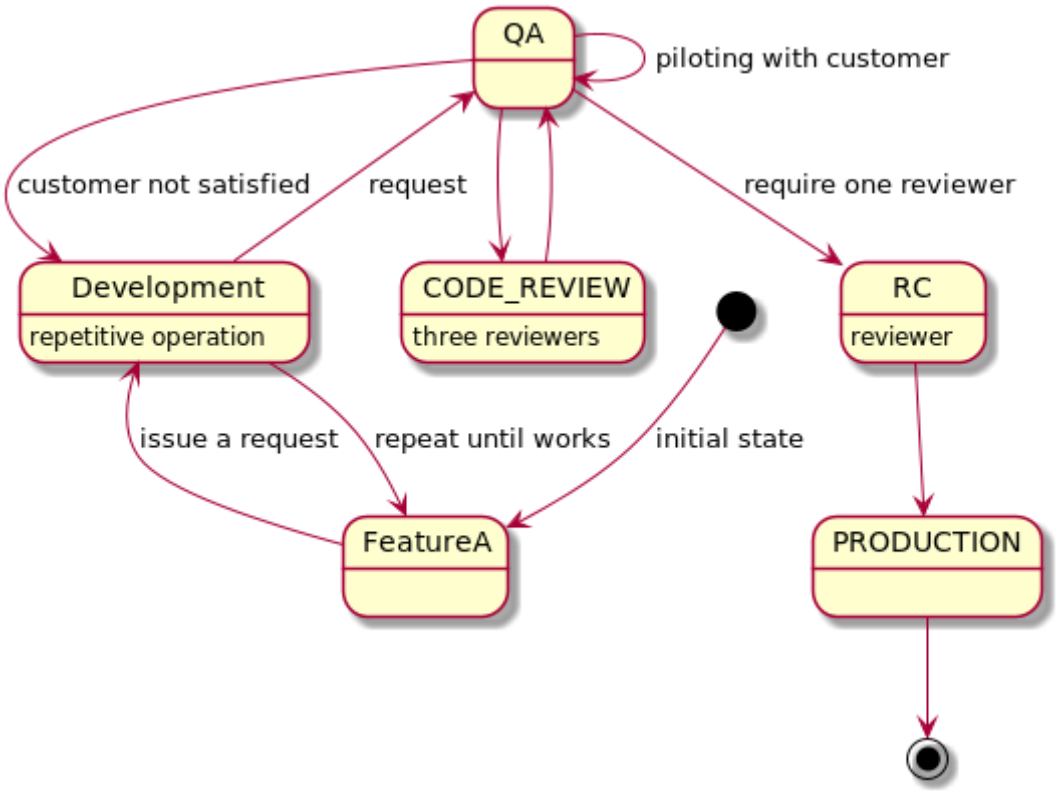


Figure 50. System validation.

For the researcher to validate the system functionality, the researcher requested to deploy the feature to quality assurance. In quality assurance, there were three senior reviewers to evaluate the code before piloting with the customer. After receiving the

approval from the reviewers, the feature resided in quality assurance and product pilot was tested with one of the Invian Oy customers. The piloted resource planning feature tested with a customer with a couple of hundred users, the piloting period was one month. After the period ended, the customer gave positive feedback about the system with some new ideas which intended to do in the future. Then, a request issued to the release candidate who requires one reviewer, after receiving the approval, the feature deployed to the release candidate container and because of the customer was satisfied with the pilot stage, Invian Oy decided to prepare for deployment of the resource planning feature to production and later gave it to the customer. For the first customer, there were about thousands of users who were using the system daily. Then, there were dozens of other home care, and self-assistant companies started using the feature with about 40000 users daily. Regards the UI, due to the company policy, any publishment to user interface that expose the APIs is not allowed.

In conclusion, the system validated through two steps, and Invian Oy QA team did the first step internally. In addition, multiple clients validated the other step.

5. DISCUSSION

This section discusses how the objective of this thesis achieved, the added values to the company, the improvements which added to the current ERP system, and finally the retrospect in the context of writing the thesis within a company environment.

5.1. Achieving the Goals Set for the Thesis

Based on the knowledge obtained from the literature review, the API was designed to fulfil the theory concerning the REST constraints and approach. All API were carefully implemented to fit in the stateless model, which is the HTTP, and with uniform interfaces to communicate between components and client-server existence, which are Tomcat container and Jersey to fulfill the server requirements and postman and web for the client. Improving the database relation through entity relation constraints was achieved with the help of MySQL workbench, which assisted in adding DB sources, reverse engineering, and plot the required tables in the ER diagram. The added constraints to the database have supported in keeping data consistency and integrity. Providing activity diagrams based on customer requirements was done and achieved through studying the needs of the customer jointly with the product owner and produces the activity diagrams. Documentation added alongside the development of the APIs, swagger used for API documentation, other documentation like sequences diagrams and activity diagrams provided. Finally, testing the implemented APIs was done using JUnit testing and based on the unit tests results, and validation of the system with one of Invian Oy clients, the goals of this thesis achieved.

Ultimately, the thesis did produce functional RESTful interfaces that fulfil the requirements of the client as well as were proven functional in test cases as well as in a live deployment.

5.2. Improving the DomaCare ERP

DomaCare ERP is designed to fit into the social health care sector. The RESTful services implemented in this thesis added the new potential possibilities to the current DomaCare ERP resource planning system to expand to more platforms such as the web, mobiles (Android, IOS, Nokia) and embedded systems. The ability to restart server is one of the benefits gained with the REST nature of the stateless design of the API. Added greater flexibility in querying information and faster requests processing through uniforms interfaces and caches. Moreover, the possibility of integration of other systems and services with the existing resources.

Currently, there are dozens of Invian Oy clients using DomaCare resource planning module on daily with about 40000 users.

5.3. Reflection

One of the difficulties that faced the researcher was studying the entities relation and understand how they operate in practice. Dealing with privacy was one of the difficulties that faced the researcher due to dealing with client sensitive information.

6. CONCLUSION

The primary objective of this thesis was to implement RESTful API for resource management in which users can plan and manage resources for home care and personal assistant services in the context of an ERP system. To achieve this goal, a conducted vast literature review including but not limited to REST and other functions. A comprehensive study to the current tables structure and draws the entities relations from that structure with adding more constraints to some part of the existing tables as an improvement achieved, this thesis also designed and implemented the APIs in the presentation layer. Furthermore, both the domain layer contains the services, validators, and business logic of the application and DAOs layer that provide connection to the database and execute crud operation was designed and implemented. Documentation was part of the implementation and swagger framework was used for documenting the API. Other documentation like class UML design, activities diagrams, and sequences diagrams were provided to accomplish this research. With jersey framework, Tomcat servlet container, JOOQ, and other tools made it possible to complete this research with great satisfaction from the management.

The API developed in this thesis was a production quality, and some part of it is already deployed to the production line and used by a tenth of thousands of Invian Oy customers daily. With the rapid growth of ERP systems, there is a huge race and competitions between companies in delivering the best ERP systems. With the rapid growth of technology, IOT considered one of the hottest topics in the century. With, there will be a lot of new ideas coming up to ERP which will significantly improve the existing systems.

7. REFERENCES

[1] Fielding R. T. and Taylor R. N. 2002. Principled Design of the Modern Web Architecture. Information and Computer Science University of California, Irvine.

[2] Perry D. E. and Wolf A. L. 1992. Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes.

[3] Fielding R. T. 2000. Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.

[4] ISO/IEC/IEEE 42010:2011 Systems and software engineering - Architecture description, 2011. Consulted April 2015 https://www.iso.org/obp/ui/#iso:std:50508:en.

[5] Garlan D. and Shaw M. 1994. An Introduction to Software Architecture. School of Computer Science, Carnegie Mellon University.

[6] Shaw M. and Clements P. 1997. A field guide to boxology: Preliminary classification of architectural styles for software systems. COMPSAC '97. Proceedings.

[7] Clements P.; Bachmann F.; Bass L.; Garlan D.; Ivers J.; Little R.; Merson P.; Nord R.; Stafford J. 2003. Documenting Software Architectures: Views and Beyond. Addison-Wesley.

[8] RFC 3986.2005. Uniform Resource Identifier (URI): Generic Syntax. Consulted April 2015 https://tools.ietf.org/html/rfc3986.

[9] RFC 7230.2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Consulted April 2015 https://tools.ietf.org/html/rfc7230.

[10] JAVA 8 API Documentation https://docs.oracle.com/javase/8/docs/api/.

[11] RFC 7159.2014. The JavaScript Object Notation (JSON) Data Interchange Format. Consulted April 2015 https://tools.ietf.org/html/rfc7159.

[12] Junit 4.12 API Documentation. Consulted April 2015 http://junit.org/javadoc/latest/index.html.

[13] Jersey 2.27 API Documentation https://jersey.github.io/documentation/latest/index.html.

[14] Tomcat 8.5 Server Documentation http://tomcat.apache.org/.

[15] MySQL 8 Server Documentation https://dev.mysql.com/doc/refman/8.0/en/.

[16] JOOQ https://www.jooq.org/.

[17] IntelliJ IDEA https://www.jetbrains.com/idea/.

[18] Martin Fowler. UML Distilled Third Edition.